SAMMENLIGNING AV NUMERISKE METODER FOR MONODOMENE- OG BIDOMENEMODELLEN

av

Else Bergene Zakariassen

MASTEROPPGAVE

for graden

Master i Anvendt matematikk og mekanikk

(Master of Science)

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

November 2010

Faculty of Mathematics and Natural Sciences
University of Oslo

FORORD

Denne oppgaven er skrevet i forbindelse med min Mastergrad i Anvendt matematikk og mekanikk ved Universitet i Oslo. Arbeidet har blitt utført ved Simula Research Laboratory i hovedsak fra januar 2010 til november 2010. Valget av oppgave ble gjort på grunnlag av min interesse for partielle differensialligninger, og spesielt løsningen av disse. I tillegg var det et meget interessant forskningfelt jeg hadde mulighet til å bli en del av. Dette har vært en slitsom, men meget lærerik periode.
Jeg vil takke

- min veileder Glenn Terje Lines som har kommet med mange gode råd gjennom hele mastergradsperioden min. Og for at du alltid har vært behjelpelig ved alle typer spørsmål og uklarheter jeg har hatt, uansett tidspunkt.
- mine foreldre for fantastisk støtte gjennom hele studietiden min.
- min mann Henrik Zakariassen for korrekturlesing og all støtte du har gitt ved oppganger og nedganger.
- og ikke minst vår herlige, lille gutt Oskar. Du er vår solstråle.

Else Bergene Zakariassen
Oslo, november 2010

Innhold

Kapittel 1

Innledning

Hjertet er et av kroppens viktigste organer, og livsnødvendig for å leve. Dens hovedfunksjon er å pumpe blod ut i kroppen. Det vil si, hjertet pumper blod ut til lungene hvor blodet tilføres oksygen og hvor karbondioksid skilles ut, og til kroppens øvrige organer hvor oksygenet i blodet avgis. For at hjertet skal kunne trekke seg sammen og pumpe blod må en elektrisk impuls bre seg gjennom hjertevevet. De elektriske impulsene som sendes ut ca 1 gang i sekundet fører til at de millionene av hjertecellene vi har trekker seg sammen omtrent samtidig. I barndommen slår hjertet opp mot 120 slag i minuttet, men denne raten vil synke etterhvert som vi vokser. Fra 18-års alderen har vi fått en hjerteslagrate på ca 60 slag i minuttet.

Før vi kan simulere den elektriske aktiviteten i hjertet og analysere resultatene, må teorien og den matematiske beskrivelsen av cellene og hjertet som en helhet forstås. Vi skal se på Monodomene- og Bidomenemodellen, som er kontinuerlige modeller som beskriver den elektriske aktiviteten i hjertet. Monodomenemodellen er en forenkling av Bidomenemodellen, og er den vi i hovedsak vil konsentreres oss om. Siden disse modellene er kontinuerlige må de dermed først diskretiseres slik at vi kan løse likningssystemene på en datamaskin.

- Kapittel 2 tar kort for seg tidligere arbeid og motivasjon for oppgaven.
- I kapittel 3 tar vi for oss hjertets anatomi, og vi beskriver elektrisk aktivitet på cellenivå og hvordan impulser beveger seg i hjertet.
- I kapittel 4 beskriver vi den matematiske strømmen i en celle, og gir tre eksempler på modeller for ionestrøm.
- Kapittel 5 beskriver hvordan elektrisk strøm kan behandles i hjertet. Vi utleder Monodomene- og Bidomenemodellen.

- Kapittel 6 tar for seg ulike numeriske metoder, og vi ser på nøyaktighet og stabilitet. Monodomene- og Bidomenemodellen blir diskretisert ved hjelp av Finite Elements metoden.
- I kapittel 7 presenteres simuleringene vi har gjort av den elektriske aktiviteten i hjertet og resultatene av disse.
- Kapittel 8 gir en oppsummering av disse resultatene.
- Tilslutt i vedlegg A blir noe av koden gjengitt.

Kapittel 2

Motivasjon og tidligere arbeid

Jakten på viten om hjertet har i de senere århundre ikke bare vært motivert av ønsket å vite mer om dette vitale organet, men i tillegg har viten om hjertet en klinisk nytte. En forståelse for hvordan hjertet fungerer kan lede til bedre teknikker for diagnostisering og behandling av hjerteproblemer. Mulighetene for å eksperimentere og gjøre kliniske studier av menneske hjertet er veldig begrenset. I tillegg vil ikke studier av dyrehjerter være tilstrekkelig, da disse skiller seg betraktelig fra menneskehjertet (hjertestørrelse, aksjonspotensialet form, varighet og restitusjon osv).

Elektriske impulser i hjertet er essensielt for at hjertet skal fungere. Siden mange hjerteproblemer er forbundet med forstyrrelser i den elektriske aktiviteten, er det knyttet stor nytteverdi til å studere den elektriske aktiviteten. Hjertearytmier, spesielt de som oppstår i ventriklene, er 3D-fenomener hvor eksperimentelle observasjoner stort sett er knyttet til overflatemålinger. Simuleringer av hjertet kan overkomme noen av disse begrensningene som finnes i eksperimentelle studier [1].

Selv om vi begynner å få stor kunnskap om hjertet, er det mange mekanismer i hjertet som ikke er fullt ut fortsått. På cellenivå er det mengder av små-skala prosesser som foregår, og forståelsen av interaksjonen av disse er begrenset. Gjennom matematiske modeller og datasimuleringer har vi muligheten til å øke vår forståelse. De fleste matematiske modeller er derimot meget komplekse, slik at det ikke er mulig å oppnå en analytisk løsning. Bidomenemodellen, utviklet i 1970-årene av Tung [2], er en meget god og velbrukt modell for å studere den elektriske flyten i hjertet. I denne oppgaven skal vi i hovedsak fokusere på en forenklet versjon av Bidomenemodellen, nemlig Monodomenemodellen. Siden Bidomene- og Monodomenemodellen består av et komplekst system av ikke-linære ODE'er (*ordinary differential equation*) og PDE'er (*partial differential equation*), kan det være vanskelig å finne effektive numeriske metoder.

Bidomenemodellen kan sees på som et koblet sett av ligninger; en elliptisk PDE, en parabolsk PDE og ODE'er. På grunn av den ikke-lineære naturen til ODE'ene så kan det være svært vanskelig å få fullt implisitte løsninger. Implisitte metoder er velkjent for å ha bedre stabilitet for stive systemer enn eksplisitte metoder, men er som regel ikke brukt grunnet kompleksiteten til systemene [3]. Operator splitting er en teknikk for å løse kompliserte likningssystemer som en sekvens av enklere likningssytemer. Dette gir muligheten for å bruke ulike numeriske metoder for hvert steg i sekvensen for å optimalisere beregningene, og for å eliminere bort komplekse avhengigheter mellom variablene [4]. Operator splitting brukes som oftest for å separere det store ikke-lineære systemet av ODE'er fra systemet av PDE'er. Dette gir oss en mulighet til å bruke implisitte eller semi-implisitte metoder for PDE-systemet. I 1968 kom Strang [5] med en type operator splittemetode. Denne Strang splittemetoden ble foreslått brukt av Qu & Garfinkel [6] for å løse Monodomenemodellen i 1999.

Modellene vi ser på er stive og av den grunn trenger vi fine romlige diskretiseringer og fine tidsdiskretiseringer. Løsningene på modellene er av den grunn simuleringsmessig dyre. Finite differences har vært populær som romlig diskretisering for å løse Bidomeneligningene [7]. Denne metoden blir som regel brukt på et regulært inndelt grid. Finite elements metode er en annen velbrukt metode [8]. Denne metoden har fordelen av at den nøyaktig kan følge den komplekse, kurvede geometrien til hjertet. For tidsdikretiseringen er eksplisitt Euler, implisitt Euler og Crank-Nicholson metoden de foretrukne valgene for numerisk metode for Bidomeneligningene. Keener & Bogar [9] hevder at Crank-Nicholson metoden er den mest effektive og nøyaktige numeriske metoden, som tillater store tidssteg uten tap av stabilitet og med mindre tap av nøyaktighet enn konkurrerende metoder. I 1985 utviklet Bank et al. [10] en 1-stegs 2-nivåers metode, TR-BDF2, hvilket vil si at metoden opererer med 2 indre steg for hvert tidssteg. Dette er en metode som innehar fordelene av 2.ordens metode backward difference formel, BDF2, uten ulempene med minnebruk. Metoden innebærer å ta et steg med trapesmetoden og deretter etterfulgt av et steg med BDF2. Trapesmetoden var et godt valg for 1. indre steg da denne metoden gir et resultat med samme nøyaktighet som BDF2 og har god stabilitet [11]. I denne oppgaven ønsker vi å sammenligne implisitt Euler, Crank-Nicholson og TR-BDF2, hvor vi blant annet ser på nøyaktigheten og CPU(*central processing unit*)-tiden disse metodene krever.

Uavhengig av tidsdiskretiseringen får vi systemer på formen $Ax = b$ som må eva-

lueres og løses for hvert tidssteg. Direkte løsere, som for eksempel Gaussian eliminering og LU dekomponering, er metoder som kommer til en eksakt løsning ved direkte matrise manipulasjoner. Iterative metoder er metoder som starter med en approksimert løsning og forsøker å redusere feilen ved hjelp av en algoritme. Konjugerte gradienters metode var en metode som viste stor forbedring ovenfor andre iterative metoder på tidspunktet den ble utviklet [4]. Videre forbedringer ble oppnådd med prekondisjonering. Den prekondisjonerte konjugerte gradienters metode (PCG) har blitt en standard teknikk for å løse store systemer [12]. Vi kommer til å bruke konjugerte gradienters metode i denne oppgaven, og vil blant annet se på nytten av prekondisjonering.

Kapittel 3

Elektrisk aktivitet i hjertet

Dette kapittelet gir en oversikt over hjertets anatomi og den elektriske aktiviteten i hjertet. I seksjon 3.1 blir det forklart hvordan hjertet er bygget opp, og hvordan blodet pumpes gjennom hjertet. I seksjon 3.2 blir det enkelt forklart hvordan cellemembranen er bygget opp og hvordan elektrisk aktivitet foregår på cellenivå. Til slutt blir det i seksjon 3.3 forklart hvordan de elektriske impulsene beveger seg i hjertet.

3.1 Hjertets anatomi

Hjertet er en muskelpumpe på størrelse med en knytteneve. Ved at hjertet driver blod gjennom kroppen, så kan cellene få tilført oksygen og næring, samtidig som cellene kan kvitte seg med avfallsstoffer. Selv om hjertet fungerer som en hul muskel som driver blod rundt i en lukket kretsløp, så er hjertemuskelvevet et veldig komplekst vev. Et forseggjort elektrisk system tillater hjertet å slå synkront og sørger for å unngå uorganisert elektrisk aktivitet som fører til arytmi [13].

Hjertet består av fire kamre, venstre og høyre atrium og venstre og høyre ventrikkel. En vegg, septium, skiller venstre og høyre del av hjertet, mens fire klaffer sørger for at blodet beveger seg i riktig retning. To av klaffene skiller venstre og høyre atrium fra henholdsvis venstre og høyre ventrikkel. De to andre klaffene skiller venstre ventrikkel fra aorta og høyre ventrikkel fra lungearterien.

Det oksygenfattige blodet fra kroppen flyter gjennom cava-venene inn i høyre atrium. Høyre atrium trekker seg deretter sammen og blodet skyves inn i høyre ventrikkel. Når blodet har kommet inn i høyre ventrikkel, trekker denne seg sammen slik at blodet pumpes ut i lungearterien som fører til lungene. Her kan blodet til-

føres nytt oksygen før det igjen skal pumpes ut i kroppen. Parallellt med at oksygenfattig blod når inn i høyre atrium, returneres oksygenrikt blod fra lungene gjennom lungevenene og inntil venstre atrium. Denne trekker seg sammen og det oksygenrike blodet skyves inn i venstre ventrikkel. Venstre ventrikkel trekker seg sammen og blodet skyves ut i aorta. Her blir blodet fordelt til arteriene som frakter blod rundt i kroppen. På vei gjennom kroppen blir blodet igjen oksygenfattig og kommer inn i hjertet nok en gang gjennom cava-venene inn til høyre atrium.

Hjerteveggen består av tre lag. Myokard er det midterste laget og er selve hjertemuskulaturen. Dette laget er ansvarlig for hjertets kontraksjonsevne. Innenfor myokard ligger endokard, en tynn, elastisk bindevevshinne dekket av epitel (overflatevev), mens epikard er en bindevevshinne som dekker utsiden av hjerteveggen. Siden ventriklene er de viktigste pumpene, er muskelveggen i ventriklene tykkere enn i atriene. Det er tykkelsen på veggen som avgjør hvor stort trykk som kan produserer når ventriklene trekker seg sammen. Den venstre ventrikkelen er den som sørger for at blod leveres til hele kroppen, og her er myokard omtrent dobbelt så tykk som for den høyre ventrikkelen.

Hjertemuskulaturen ser ut som tverrstripet skjellettmuskulatur, men oppfører seg som glatt muskulatur, hvilket vil si at den ikke er viljestyrt. Hver hjertemuskelcelle, også kalt muskelfiber, er bygget opp av sakromeer som består av tykke (myosin) og tynne (aktin) filamenter. Kontraksjonen av hjertemuskulatur skyldes glidebevegelser av filamentene i forhold til hverandre. Disse filamentene er ordnet i et mønster som repeteres gjennom hele muskelen.

3.2 Elektrisk aktivitet på cellenivå

Alle celler i kroppen er omgitt av en cellemembran. Denne membranen består hovedsaklig av to tynne lag av fosforlipider som spontant arrangeres slik at de hydrofobe (vann-redde) halene ligger mot hverandre, mens de hydrofile hodene vendes utover, mot innsiden og utsiden av cellen. På denne måten dannes et dobbeltlag som er med på å kontrollere transport av substrater gjennom membranen. De fettløselige substratene kan diffundere gjennom membranen, mens de vannløselige blir hindret. Cellemembranen er med andre ord semipermeabel.

Både innenfor og utenfor cellen finnes konsentrasjoner av blant annet kalium(K^+)-, kalsium(Ca^+)- og natrium(Na^+)-ioner. Konsentrasjonen av ionene er ulik på cellen innsiden og utside, hvilket fører til at vi får en forskjell i elektrisk ladning over

cellemembranen. Denne forskjellen i potensiale kalles **transmembranpotensiale**. Permabiliteten til cellemembranen kan endres ved å åpne og lukke ionekanaler som er spesifikke for de ulike ionetypene, og endrer dermed strømmen av ioner over cellemembranen.

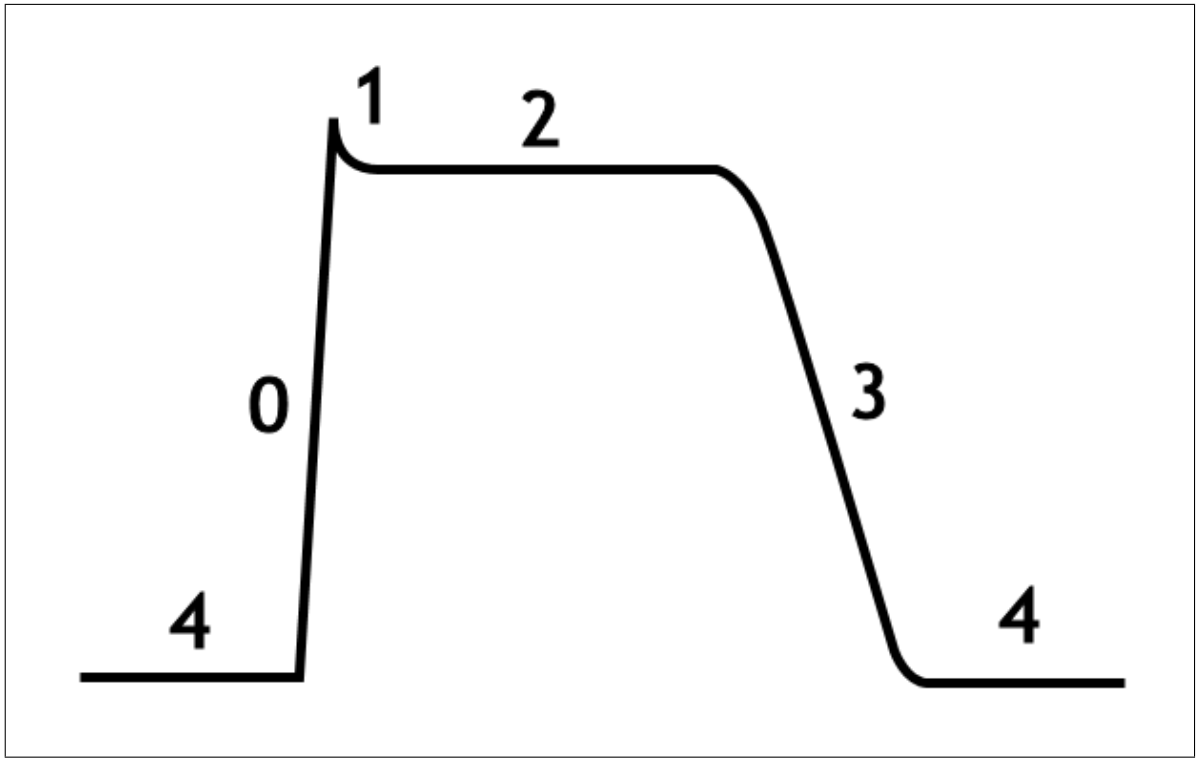
3.2.1 Hvilepotensial

I likhet med alle andre celler i kroppen, har hjertemuskelceller en større konsentrasjon av kaliumioner inne i cellen enn like utenfor. Det motsatte gjelder for natrium- og kalsiumioner, hvor konsentrasjonen altså er høyere like utenfor cellen enn innenfor. Cellemembranen har høy permeabilitet for kalium, hvilket fører til en høy nettostrøm av K^+-ioner ut av cellen drevet av konsentrasjonsgradienten. Membranen er derimot ikke like permeabel for Na^+, men permeabiliteten til Na^+ er allikevel ikke null. Dermed vil Na^+, på grunn av både konsentrasjonsgradienten (høyere Na^+ konsentrasjon på utsiden) og den elektriske gradienten (cellen er negativ på innsiden), diffundere inn i cellen. For å kunne opprettholde potensialforskjellen pumpes K^+ inn i cellen og Na^+ ut av cellen ved hjelp av et protein i membranen, kalium-natrium pumpen. På denne måten får vi det som kalles **hvilepotensiale**, som til hjerteceller er på ca -90mV [14]

3.2.2 Aksjonspotensial

Eksiterbare celler, som hjertemuskelcellene, er celler som har evnen til å genere endringer i transmembranpotensialet. Hvis en elektrisk impuls åpner en kaliumkanal og vi får en nettostrøm av K^+-ioner ut av cellen, vil transmembranpotensialet bli mer negativ. En slik økning i elektrisk gradient over membranen er kalt **hyperpolarisering**. Hvis det er en natriumkanal som åpnes på grunn av en impuls og vi får en nettostrøm av Na^+-ioner inn i cellen, vil transmembranpotensialet bli mindre negativ. En slik reduksjon i elektrisk gradient er kalt **depolarisering**. Er impulsen sterk nok og når et visst nivå kalt **terskelpotensialet**, vil et aksjonspotensiale igangsettes. Når en Na^+-port åpnes vil transmembranpotensialet stige, hvilket fører til at flere Na^+-porter åpnes. Vi får med andre ord en kjedereaksjon. Størrelsen på aksjonspotensialet er uavhengig av størrelsen på impulsen, så fremt impulsen er sterk nok til å nå terskelen. Aksjonspotensialets forløp i hjertemuskelcellene er delt i 5 faser [14], se figur 3.1.

- fase 4 - hviletilstand: Både natrium- og kaliumkanalen er tilnærmet lukket. Vi har liten flyt av Na^+-ioner og K^+-ioner over membranen. Transmembranpotensialet er på omtrent -90mV.



Figur 3.1: Hjertets aksjonspotensiale

- fase 0 - depolarisering: Aktiveringsportene i natriumkanalene åpnes, slik at Na^+-ioner flyter inn i cellen. Potensialet øker.

- fase 1 - tidlig repolarisering: K^+-ioner flyter ut av cellen. Potensialet blir mer negativt.

- fase 2 - platå fase: Samtidig som K^+-ionene strømmer ut av cellen, er det en strøm av Ca^{2+} inn i cellen. Dermed beholder cellen samme transmembranpotensiale.

- fase 3 - repolarisering: K^+-ionene strømmer fortsatt ut av cellen, og potensialet blir mer negativt.

- fase 4 - tilbake til hviletilstand: Natrium-kaliumpumpen pumper ut Na^+-ioner som kom inn i cellen under depolariseringen, og pumper inn K^+-ioner som strømmet ut av cellen under repolariseringen. Hvilepotensialet oppnås på nytt.

3.3 Elektriske impulser i hjertet

3.3.1 Aksjonspotensialets propagenering

Et område av hjertet kalt sinoatrial (SA) node opprettholder hjertets pumperytme ved å sette frekvensen på hvor ofte hjertemuskelcellene skal trekke seg sammen. Denne SA noden fungerer som hjertets pacemaker, og generer et aksjonspotensial som propagerer gjennom hjertet. SA noden er lokalisert i veggen ved høyre atrium, nær punktet hvor superior vena-cava entrer hjertet. Den generer elektriske impulser ganske likt de nervecellene produserer. Hver celle i hjertet er en selvstendig enhet, men de er også bundet sammen gjennom gap-junctions, små porer i membranen. Dermed kan hjertemuskelen sees på som et **syncytium**, den kan med andre ord oppfattes som en kontinuerlig, komplisert celle. Impulser påført til en hvilken som helst del av hjertemuskelen vil resultere i en kontraksjon av hele muskelen. En impuls fra SA noden vil med andre ord spre seg hurtig gjennom veggen i atrium slik at cellene trekker seg sammen i harmoni. Arteriene er separert fra ventriklene av et ikke-konduktivt lag. Dermed kan ikke impulsene fritt passere inn i ventriklene, men må passere gjennom et punkt kalt AV noden. Her blir impulsen forsinket med omtrent 0.1 sek, hvilket sørger for at atrium trekker seg sammen først og får tømt seg fullstendig før ventriklene trekker seg sammen [15]. I motsetning til SA noden sprer ikke AV noden impulser i alle retninger, men går over i His' bunt. His' bunt deler seg i to forgreininger, en til venstre og en til høyre ventrikkel. De to forgreningene av His' bunt står i forbindelse med Purkinjefibrene, som igjen står i forbindelse med muskelcellene i ventriklene. Impulsene som kommer fra AV noden vil spre seg hurtig gjennom de to forgreiningene i His' bunt og Purkinjefibrene. Etter Purkinjefibrene går impulsene så fra celle til celle gjennom muskelvevet i ventriklene. Siden impulsene ledes raskt gjennom His' bunt og Purkinjefibrene så begynner sammentrekningen av en stor del av hjerteveggen samtidig.

3.3.2 Anisotropi

Hjertet er bygget opp av muskelfibre, og muskelfibrenes retning har betydning for hvordan aksjonspotensialet brer seg i hjertet. Fiberretningen varierer gjennom hjertemuskelen, og gir ulik ledningsevne, hvor konduktiviteten er høyere i retning av fibrene enn den er på tvers av fibrene. Det at fibrene har ulik ledingsevne avhengig av retningen på fibrene kalles en anisotropisk egenskap. Anisotropi er vanskeligere å modellere, og ofte forenkles problemet ved å anta isotropi, det vil si at ledningsevnen er lik uavnhengig av retningen på fibrene.

Kapittel 4

Matematisk beskrivelse av cellene

I dette kapittelet skal vi komme frem til en modell for cellemembranen. For å kunne formulere en modell trenger vi kunskaper om transmembranpotensiale, strøm og fluks av ioner over cellemembranen, hvilket blir beskrevet i seksjon 4.1, seksjon 4.2 og seksjon 4.3. I seksjon 4.4 forklares det kort om ionestrøm, mens i seksjon 4.5 gis det tre eksempler på modeller for ionestrøm. Det er Aliev-Panfilovs modell for ionestrøm vi i hovedsak skal bruke videre i oppgaven. I tillegg vil vi se på en modell av ten Tusscher et al.

4.1 Transmembranpotensiale

Alle celler har en ladningsforskjell over cellemembranen, transmembranpotensiale, som skyldes forskjell i konsentrasjonen av positive og negative ioner. Dette potensialet kan defineres som

$$V = u_i - u_e \ ,$$

hvor u_i er det intracellulære potensialet og u_e er det ekstracellulære potensialet.

4.2 Transmembranstrøm

Cellemembranen oppfører seg både som en motstand og en kondensator. Den leder strøm gjennom ionekanaler og den separerer ladning. Transmembranstrømmen I_m er summen av kapasitansstrømmen I_c og ionestrømmen I_{ion}.

$$I_m = I_c + I_{ion} \ .$$

Kapasitansen er definert som

$$C_m = \frac{Q}{V} \ ,$$

hvor Q er ladningsforskjell over kondensatoren og V er potensialet.

$$V = \frac{1}{C_m} \cdot Q \ .$$

Hvis ladningen endres i tidsrommet Δt får vi også en endring i potensialet

$$\frac{\Delta V}{\Delta t} = \frac{1}{C_m} \frac{\Delta Q}{\Delta t} \ .$$

Strøm er definert som

$$I = \lim_{\Delta t \to 0} \frac{\Delta Q}{\Delta t} \ .$$

Dermed får vi kapasitiv strøm I_c når $\Delta t \to 0$:

$$C_m \frac{dV}{dt} = I_c \ .$$

I_{ion} representerer en ikke-lineær funksjon av V og t siden membranens resistans endrer seg. Dermed kan transmembranstrømmem pr arealenhet uttrykkes

$$I_m(V,t) = C_m \frac{dV}{dt} + I_{ion}(V,t) \ .$$

For å kunne uttrykke transmembranstrømmen per volumenhet må vi innføre en konstant χ. Denne konstanten uttrykker forholdet mellom celleareal og cellevolum. Endelig modell for I_m blir da:

$$I_m(V,t) = \chi \left(C_m \frac{dV}{dt} + I_{ion}(V,t) \right) \ . \tag{4.1}$$

4.3 Fluks av ioner

Strømmen av ioner over membranen er drevet av konsentrasjonsforskjellen av ioner og den elektriske gradienten over membranen. Bidraget fra den elektriske gradienten til fluks er gitt ved Planck's ligning [16]

$$J_E = -m \frac{z}{|z|} c \nabla \phi \ ,$$

hvor m er friksjon, c er ionekonsentrasjon, $\frac{z}{|z|}$ er fortegnet på ioneladningen og ϕ er potensialet, slik at $\nabla \phi$ er det elektriske feltet.

Bidraget fra konsentrasjonsgradienten til fluks er gitt ved Flick's lov

$$J_D = -D\nabla c \ ,$$

hvor D er er Fick's diffusjonskonstant og c er ionekonsentrasjon.

$$D = m\frac{RT}{|Z| \cdot F} \ ,$$

med R som er gasskonstanten, F som er Farday konstant, T som er temperatur . For en fungerende celle har vi normalt både forskjeller i elektrisk potensial og ionekonsentrasjoner over membranen, og den totale fluksen er gitt ved

$$J = J_E + J_D \ .$$

Når vi kombinerer fluksen gitt av den elektriske gradienten og fluksen gitt av konsentrasjonsgradienten får vi Nernst-Planck ligning

$$J = -D\left(\nabla c + \frac{zF}{RT}c\nabla\phi\right) \ .$$

Fluksen J er null når vi er i likevekt. I 1D får vår vi da

$$-D\left(\frac{dc}{dx} + \frac{zF}{RT}c\frac{d\phi}{dx}\right) = 0 \ ,$$

slik at

$$\frac{1}{c}\frac{dc}{dx} + \frac{zF}{RT}\frac{d\phi}{dx} = 0 \ .$$

Om vi antar at cellemembranen går fra $x = 0$ på innsiden til $x = L$ på utsiden, kan vi integrere fra $x = 0$ til $x = L$ og vi får

$$\ln(c)|_{c(0)}^{c(L)} = \frac{zF}{RT}(\phi(0) - \phi(L)) \ .$$

Vi lar $\phi(0) = \phi_i$, $\phi(L) = \phi_e$, $c(0) = c_i$ og $c(L) = c_e$ hvor i og e angir størrelsen på henholdsvis innsiden og utsiden av cellemembranen. Potensialet ved likevekt er gitt ved $V_{eq} = \phi_i - \phi_e$

$$V_{eq} = \frac{RT}{zF}\ln\left(\frac{c_e}{c_i}\right) \ , \tag{4.2}$$

hvilket er Nernst potensial. Denne uttrykker likevektspotensialet til et gitt ion over cellemembranen.

4.4 Ionestrøm over cellemembranen

Det finnes mange modeller for ionestrømmen. Ofte brukes det at strømmen over cellemembranen er en lineær funksjon av transmembranpotensialet

$$I_{ion}(V,t) = g(V - V_{eq}) \ ,$$

hvor $g = 1/r$ er konduktansen som beskriver ionekanalens evne til å lede en bestemt type ioner gjennom membranen og r er resistansen. g er ikke nødvendigvis en konstant. Denne type modell beskriver bare de store linjene for celleoppførsel, men det fulle potensialet for simuleringene kan bare nåes ved å bruke mer sofistikerte modeller, slik som FitzHugh-Nagumo modellen [16].

4.5 Noen cellemodeller

4.5.1 FitzHugh-Nagumo modell

FitzHugh-Nagumo(FHN) er en forenkling av Hodgkin-Huxley modellen [17] som ble dannet i 1952. Begge modellene beskriver den elektriske aktiviteten i den enkelte celle, med andre ord beskriver den hva som skjer med transmembranpotensialet og ionestrømmer i cellen. Fra Hodgkin-Huxley modellen til FHN modellen har vi fått en reduksjon fra fire til to variabler, v og w, hvor v beskriver potensialet på overflaten og w modellerer natrium- og kalsiumkanalene.
Den orginale FitzHugh-Nagumo modellen [18] er gitt av

$$\begin{aligned} \frac{dv}{dt} &= c_1 v(v-a)(1-v) - c_2 w + i_{app} \\ \frac{dw}{dt} &= b(v - c_3 w) \ , \end{aligned}$$

hvor a, b, c_1, c_2 og c_3 er gitte parametere, og i_{app} er tilført strøm som trigger aksjonspotensialet. I den orginale modellen var parameterene gitt som $a = 0.13$, $b = 0.013$, $c_1 = 0.26$, $c_2 = 0.1$ og $c_3 = 1.0$ [16].

Denne orginale FHN-modellen har noen begrensninger, da den blandt annet gir en hyperpolarisering av cellen i repolariseringsfasen (fase 3). Rogers & McCulloch [19] kom med en modifisering for å overkomme dette problemet i 1994. Dermed

blir modellen

$$\frac{dv}{dt} = c_1 v(v-a)(1-v) - c_2 wv + i_{app}$$
$$\frac{dw}{dt} = b(v - c_3 w) \ .$$

For å få realistiske modeller av transmembranpotensialet er det nødvendig å endre variablene i modellen.

$$V = v_{amp} v + v_{rest}$$
$$W = v_{amp} w \ ,$$

hvor v_{amp} er amplituden til aksjonspotensialet, v_{rest} er potensialet i hviletilstand (fase 4) og v_{peak} er maksimum potensialet.
Setter inn for de nye variablene i den modifiserte modellen og får

$$\frac{dV}{dt} = \frac{c_1}{v_{amp}^2}(V - v_{rest})(V - v_{th})(v_{peak} - V) - \frac{c_2}{v_{amp}}(V - v_{rest})W + I_{app}$$
$$\frac{dW}{dt} = b(V - v_{rest} - c_3 w) \ ,$$

hvor I_{app} er gitt som $I_{app} = v_{amp} i_{app}$ og terskelverdien er definert som $v_{th} = v_{rest} + a v_{amp}$.

4.5.2 Aliev-Panfilov modell

FHN-modellen feiler ved beskrivelse av flere kvantitative parametere av hjertevev, slik som blant annet formen til aksjonspotensialet. Mange forbedringer av modellen har blitt gjennomført, men for kvalitativ analyse er de, etter Aliev og Panfilov sin mening, vanskelige å bruke på grunn av sin kompliserte matematiske representasjon [20]. Dermed kom Aliev og Panfilov med en enkel modell som simulerer restitusjonsegenskaper, gir en god representasjon av formen på aksjonspotensialet og som effektivt kan brukes i numeriske simuleringer.

$$\frac{\partial u}{\partial t} = -ku(u-a)(u-1) - uv$$
$$\frac{\partial v}{\partial t} = \epsilon(u,v)(-v - ku(u-a-1)) \ ,$$

hvor $\epsilon(u,v) = \epsilon_0 + \frac{\mu_1 v}{u+\mu_2}$, $k = 8$, $a = 0.15$, $\epsilon_0 = 0.002$, μ_1 og μ_2 er parametere som skal fikseres etter eget valg. u, v og t er dimensjonsløse variabler.

4.5.3 TNNP modell

En annen modell vi skal se nærmere på, er en modell av ten Tusscher et al. [1]. Denne modellen inkluderer et høyt nivå av elektrofysiske detaljer og kan dermed reprodusere detaljerte egenskaper til menneskelig ventrikkelceller, blant annet viktige ione-strømmer, aksjonspotensialets varighet og hvile, og viktige egenskaper ved bølgepropagering i menneskelig ventrikkelvev. Modellen er basert på eksperimentelle data av noen viktige ione-strømmer.

Den elektrofysiologiske oppførselen til en celle kan beskrives med følgende ligning:

$$\frac{\partial V}{\partial t} = -\frac{I_{ion} + I_{app}}{C_m} \ ,$$

hvor C_m er cellens kapasitans per enhets overflateareal, I_{app} er den tilførte strømmen, mens I_{ion} er summen av alle de transmembrane strømmene gitt av følgende ligning:

$$I_{ion} = I_{Na} + I_{K1} + I_{to} + I_{Kr} + I_{Ks} + I_{CaL} + I_{NaCa} + I_{NaK} + I_{pCa} + I_{pK} + I_{bCa} + I_{bNa}$$

TNNP modell kan videre beskrives som gatingligninger på formen:

$$j = \frac{1}{\tau_j}(f_\infty - f) \ ,$$

og ligninger som beskriver endringen av ionekonsentrasjon over tid, for eksempel denne for kalium:

$$\frac{dK_i}{dt} = -\frac{I_{K1} + I_{to} + I_{Kr} + I_{Ks} - 2I_{NaK} + I_{pK} + I_{app} - I_{ax}}{V_c F} \ .$$

Modellen besår av 19 state variable.

Kapittel 5

Matematisk beskrivelse av hjertet

I dette kapittelet beskrives Bidomenemodellen og Monodomenemodellen. Seksjon 5.1 tar for seg Bidomenemodellen, mens seksjon 5.2 tar for seg Monodomenemodellen.

5.1 Bidomenemodellen

Den elektriske aktiviteten i hjertet er beskrevet ved det fullt koblede systemet av ordinære og partielle differensialligninger, kalt Bidomenemodellen

$$\begin{aligned}
\frac{\partial s}{\partial t} &= F(s, V, t) && x \in H \qquad (5.1)\\
\chi\left(C_m \frac{\partial V}{\partial t} + I_{ion}(V,t) - I_{app}\right) &= \nabla \cdot (M_i \nabla V) + \nabla \cdot (M_i \nabla u_e) && x \in H\\
\nabla \cdot (M_i \nabla V) &= -\nabla \cdot ((M_e + M_i)\nabla u_e) && x \in H\\
V(x,0) &= I && x \in H\\
n \cdot (M_i \nabla V + M_i \nabla u_e) &= 0 && x \in \partial H\\
n \cdot (M_e \nabla u_e) &= 0 && x \in \partial H
\end{aligned}$$

hvor V er det transmembrane potensialet, u_e det ekstracellulære potensialet, s er en tilstandsvektor for cellemodell, mens M_i og M_e er henholdsvis de intra - og ekstracellulære konduktivitetstensorene. H er hjertevolumet.

5.1.1 Anisotropi

Ledningsevnen til hjertevev er anistropisk. Dette følger av at hjertemuskelen består av fibre som ligger i ulike retninger. Konduktiviteten vil variere avhengig av ret-

ningen på fibrene, og dermed vil konduktivitetstensorene varierer gjennom hjertet. Den lokale konduktivitetstensoren er definert som [16]

$$M^* = \begin{bmatrix} \sigma_l & 0 & 0 \\ 0 & \sigma_t & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}$$

hvor σ_l er konduktivitet langs fiberretningen, og σ_t og σ_n er konduktivitet på tvers av fiberretningen. Vi kan transformere den lokale konduktivitetstensoren til de to globale konduktivtstensorene M_i og M_e slik:

$$M_i = AM_i^*A^T$$

$$M_e = AM_e^*A^T$$

hvor $A = (a_l \mid a_t \mid a_n)$ er en matrise bestående av tre ortogonale enhets-kolonnevektorer som kan variere med posisjon.

5.1.2 Utledning av modellen

Vi starter med å anta at strømmen I i vevet er ohmsk

$$I = -g\nabla u \ ,$$

der $g = 1/r$ er konduktansen, r er resistansen og u er spenningen. For å få strømstyrken i et punkt kan vi anta at motstanden ligger langs x-aksen, dividere med Δx og la $\Delta x \to 0$

$$J = -g\frac{du}{dx} \ .$$

Generaliserer og får i flere dimensjoner

$$J = -M\nabla u \ , \tag{5.2}$$

hvor M er konduktivitetstensoren.
Cellemembranen oppfører seg som en isolator mellom to domener, og kan dermed skille ladninger. Imidlertid, siden den er så tynn, vil oppsamling av ladning på den ene siden av membranen umiddelbart tiltrekke seg motsatt ladning fra den andre siden. Dermed vil den totale ladningsforflytningen over membranen være null

$$\frac{\partial}{\partial t}(q_i + q_e) = 0 \ , \tag{5.3}$$

hvor q_i er den intracellulære ladningen og q_e er den ekstracellulære ladningen. Netto strøm i et punkt i hvert domene er lik endringen i ladning og ionestrømmen i det punktet

$$-\nabla \cdot J_i = \frac{\partial q_i}{\partial t} + \chi I_{ion} \tag{5.4}$$

$$-\nabla \cdot J_e = \frac{\partial q_e}{\partial t} - \chi I_{ion} \tag{5.5}$$

Kombinerer vi Ligning (5.4) og Ligning (5.5) med Ligning (5.3) får vi

$$\nabla \cdot J_i + \nabla \cdot J_e = 0 \ , \tag{5.6}$$

hvilket vil si at totalstrømnmen er bevart.
Vi kan kombinere Ligning (5.6) med Ligning (5.2) og få

$$\nabla \cdot (M_i \nabla u_i + M_e \nabla u_e) = 0 \ . \tag{5.7}$$

Transmembranstrømmen er definert som $I_m = -\nabla J_i$. Dette er strøm som går fra det intracellulære rommet til det ekstracellulære rommet. Siden $J_i = -M_i \nabla u_i$ får vi

$$I_m = -\nabla \cdot J_i = \nabla \cdot (M_i \nabla u_i) \ . \tag{5.8}$$

Ved å kombinere Ligning (5.7) med Ligning (5.8) får vi systemet

$$\begin{aligned} I_m &= \nabla \cdot (M_i \nabla u_i) \\ \nabla \cdot (M_i \nabla u_i) &= -\nabla \cdot (M_e \nabla u_e) \ . \end{aligned} \tag{5.9}$$

Fra Ligning (4.1) har vi at I_m pr volumenhet kan uttrykkes på følgende måte

$$I_m(V,t) = \chi \left(C_m \frac{\partial V}{\partial t} + I_{ion}(V,t) \right) \ .$$

I tillegg til ionestrømmen og den kapasitive strømmen kan vi ha en tilført strøm I_{app}. Dette gir da

$$\begin{aligned} \chi \left(C_m \frac{\partial V}{\partial t} + I_{ion}(V,t) - I_{app} \right) &= \nabla \cdot (M_i \nabla u_i) \\ \nabla \cdot (M_i \nabla u_i) &= -\nabla \cdot (M_e \nabla u_e) \ . \end{aligned} \tag{5.10}$$

Ved å eliminere bort u_i fra Ligningssystem (5.10) får vi

$$\chi\left(C_m\frac{\partial V}{\partial t}+I_{ion}(V,t)-I_{app}\right)=\nabla\cdot(M_i\nabla u_i)+(\nabla\cdot(M_i\nabla u_e)-\nabla\cdot(M_i\nabla u_e))\ . \quad (5.11)$$

$$\nabla\cdot(M_i\nabla u_i)-\nabla\cdot(M_i\nabla u_e)=-\nabla\cdot(M_e\nabla u_e)-\nabla\cdot(M_i\nabla u_e)\ .$$

Siden $V=u_i-u_e$ får vi

$$\chi\left(C_m\frac{\partial V}{\partial t}+I_{ion}(V,t)-I_{app}\right)=\nabla\cdot(M_i\nabla V)+\nabla\cdot(M_i\nabla u_e) \quad (5.12)$$

$$\nabla\cdot(M_i\nabla V)=-\nabla\cdot((M_e+M_i)\nabla u_e)\ ,$$

hvilket er Bidomeneligningene for modellering av elektrisk aktivitet i hjertet.

5.1.3 Initialbetingelser og randkrav

Verdier for de lokale konduktivitene σ_l og σ_t blir gitt i tabell 5.1. I tillegg spesifiserer tabellen overflate-til-volum ratioen χ og membrankapasitansen C_m. Så fremt inget annet er oppgitt, vil disse verdiene bli brukt.

C_m	1.0 μF/cm^2
χ	2000 cm^{-1}
σ_l^i	3.0 mS/cm
σ_t^i	0.3 mS/cm
σ_l^e	2.0 mS/cm
σ_t^e	1.4 mS/cm

Tabell 5.1: Verdier for parametere i Bidomene- og Monodomenemodellen

For startverdi er en mulighet å anta at vevet initielt har hvilepotensialet V_{rest}, hvilket betyr at vevet ikke er aktivt. Altså

$$V(x,0)=V_{rest} \qquad \text{i } H \quad (5.13)$$

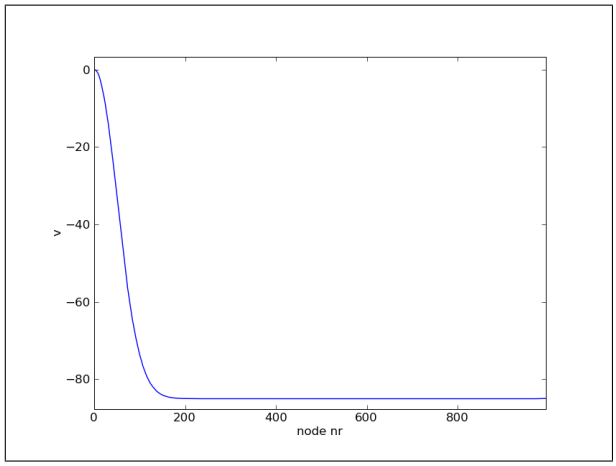
Dermed må et elektrisk signal initieres med en tilført strøm I_{app}. Denne muligheten vil benyttes når vi ser på TNNP cellemodell, se seksjon 4.5.3 og seksjon 7.3.

En annen mulighet, som har blitt benyttet i store deler av denne oppgaven, er å se på en propagerende vegg over domenet. Vi sikrer at systemet aktiveres ved at $v > v_{th}$(terskelpotensialet) for en liten region Ω_i, mens det for stort sett resten av

domenet er $v = v_{rest}$. Over tid vil verdien av v i Ω_i stige raskt mot v_{max}. Ved tilstrekkelig høy konduktivitet vil diffusjon bringe v over v_{th} i en liten region omkring Ω_i, og fortsette til v har nådd v_{max} i hele domenet. Initialbetingelsen vår blir da

$$v(x, 0) = -85(1 - e^{-200x^2}) \, . \tag{5.14}$$

Figur 5.1 viser verdiene for transmembranpotensialet i 1D, hvor vi kan se at v i hovedsak er lik hvilepotensialet.



Figur 5.1: Verdier for v i 1D

Randkrav som skal brukes er

$$n \cdot (M_i \nabla V + M_i \nabla u_e) = 0 \qquad \text{på } \partial H \tag{5.15}$$

$$n \cdot (M_e \nabla u_e) = 0 \qquad \text{på } \partial H \tag{5.16}$$

5.2 Monodomenemodellen

Bidomenemodellen er et system av ordinære og partielle differensialligninger som er vanskelig å løse og å analysere. Ved å anta uniform anisotropi kan vi få en forenkling av den orginale Bidomenemodellen. Monodomenemodellene gir oss en forenklet modell som er nyttig for analysering og enkle studier av den elektriske hjerteaktiviteten. Den har imidlertid noen svakheter fysiolgisk på grunn av antagelsen om uniform anisotropi, slik at for realistiske simuleringer av en del fysiologiske fenomener kreves det bruk av Bidomenemodellen [16, jmf s.31].

Monodomenemodellen er gitt ved

$$\begin{aligned} \frac{\partial s}{\partial t} &= F(s,V,t) && x \in H \\ \chi\left(C_m\frac{\partial V}{\partial t} + I_{ion}(V,t) - I_{app}\right) &= \nabla\cdot M\nabla V && x \in H \\ V(x,0) &= I && x \in H \\ n^T M\nabla V &= 0 && x \in \partial H \end{aligned} \tag{5.17}$$

5.2.1 Utledning av modellen

For å utlede Monodomenemodellen fra Bidomenemodellen antar vi uniform anisotropi, altså $M_e = \lambda M_i$ hvor λ er en konstant skalar. Dermed kan vi elimenere M_e fra tredje ligning i System (5.1)

$$\begin{aligned} \nabla\cdot(M_i\nabla V) &= -\nabla\cdot((M_i+\lambda M_i)\nabla u_e) \\ &= -(1+\lambda)(\nabla\cdot(M_i\nabla u_e))\ . \end{aligned}$$

$$\nabla\cdot(M_i\nabla u_e) = -\frac{1}{1+\lambda}(\nabla\cdot(M_i\nabla V))\ . \tag{5.18}$$

u_e kan nå elimineres fra andre ligning i System (5.1). Setter inn Ligning (5.18) og får

$$\chi\left(C_m\frac{\partial V}{\partial t} + I_{ion}(V,t) - I_{app}\right) = \frac{\lambda}{1+\lambda}(\nabla\cdot(M_i\nabla V))\ .$$

Vi innfører $M = \frac{\lambda}{1+\lambda}M_i$

$$\chi\left(C_m\frac{\partial V}{\partial t} + I_{ion}(V,t) - I_{app}\right) = \nabla\cdot(M\nabla V)\ . \tag{5.19}$$

På denne måten er det mulig å finne transmembranpotensialet ved å løse Ligning (5.19), og eventuelt Ligning (5.18) for å finne det ekstracellulære potensialet.

Hvis vi setter I_{ion} = I_{app} = 0 og skalerer Ligning (5.19) får vi et spesialtilfelle av Monodomeneproblemet.

$$\frac{\partial V}{\partial t} = \nabla\cdot(M\nabla V)\ . \tag{5.20}$$

hvilket er varmeledningsligningen.

5.2.2 Initialbetingelser og randkrav

Verdier for de lokale konduktivitene σ_l og σ_t, samt verdier for overflate-til-volum ratioen χ og membrankapasitansen C_m, ble gitt i tabell 5.1.

Som for Bidomenemodellen settes initialbetingelsen slik

$$v(x,0) = -85(1 - e^{-200x^2}) \, . \tag{5.21}$$

Randkravet er

$$n^T M \nabla V = 0 \qquad \text{på } \partial H \tag{5.22}$$

Kapittel 6

Numeriske metoder

Mange differensialligninger kan ikke løses analytisk, men vi kan løse dem tilfredsstillende nok ved en approksimasjon av løsningen. Siden det er ønskelig med en god numerisk approksimasjon av løsningen, er det viktig å velge tilfredsstillende numeriske metoder. I seksjon 6.1 vises det eksempler på ulike numeriske skjemaer og vi ser på nøyaktigheten og stabiliteten av disse. I seksjon 6.2 ser vi på to ulike splittemetoder for kompliserte problemer. Seksjon 6.3 tar for seg diskretiseringen av Bidomenemodellen og Monodomenemodellen. I tillegg gis det i seksjon 6.4 et eksempel på en ODE-løser som skal brukes videre i oppgaven, og i seksjon 6.5 forklares den iterative løseren som brukes for å løse lineære likningssystemer.

6.1 Numeriske skjema for tidsdiskretisering

Når vi løser en differensialligning numerisk er det ønskelig at feilen mellom den eksakte løsningen og den numeriske løsningen blir så liten som mulig, samtidig som det går raskest mulig. I tillegg til at løsningen bør være nøyaktig, er vi avhengig av at de numeriske metodene vi bruker er stabile. Vi skal snakke om to ulike typer stabilitet, A-stabilitet og L-stabilitet, hvor L-stabilitet er svært attraktivt når vi skal jobbe med veldig stive systemer.

6.1.1 Implisitt Euler metode

Nøyaktighet

Implisitt Euler er en 1.ordens metode, hvilket betyr at når tidssteget halveres vil i tillegg feilen halveres. Det vil si at vi ofte må ha et veldig lite tidssteg for at vår numeriske løsning skal nærme seg den eksakte løsningen. Ved hjelp av taylor rekkeutvikling vises det her hvorfor implisitt Euler er en 1.ordens metode.

Vi har en differensialligning

$$\frac{dy(t)}{dt} = f(t, y(t)) \quad \in [a, b]$$

$$y(0) = y_0 \ .$$

Vi deler intervallet [a,b] i N deler med tidssteg $\triangle t = \frac{(b-a)}{N}$ og $t_i = a + i \triangle t$ for $i = 0, 1...N$. En numerisk løsning gitt ved implisitt Euler metoden er

$$\frac{y_{n+1} - y_n}{\triangle t} = f(t_{n+1}, y_{n+1}) \ .$$

$$\begin{aligned} y_{n+1} &= y_n + \triangle t f(t_{n+1}, y_{n+1}) = y_n + y'_n \\ &= \hat{y}(t) \ . \end{aligned} \tag{6.1}$$

Denne metoden er en implisitt metode siden den evaluerer f i (t_{n+1}, y_{n+1})

Program 1 viser hvordan vi kan løse problemet (6.3) med implisitt Euler.

Program 1 Implisitt euler på et testproblem

```
T = 10
dt = 0.25
N = int(T/dt)
lambda1 = 0.1
y = 1
for i in range (0,N):
    y = 1.0/(1-lambda1*dt)*y
print 'y = ', y
```

For å finne feilen for implisitt Euler metoden kan vi ta en Taylor rekkeutvikling for y rundt t_{n+1}

$$y(t) = y(t_{n+1}) + \frac{dy(t_{n+1})}{dt}(t - t_{n+1}) + \frac{1}{2}\frac{d^2y(t_{n+1})}{dt^2}(t - t_{n+1})^2 \ .$$

Evaluerer i t_n

$$\begin{aligned} y(t_n) &= y(t_{n+1}) - \Delta t \frac{dy(t_{n+1})}{dt} + O(\Delta t^2) \\ &= y(t_{n+1}) - \Delta t f((t_{n+1}, y_{n+1}) + O(\Delta t^2) \ . \end{aligned}$$

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \Delta t f(t_{n+1}, y_{n+1}) - O(\Delta t^2) \\ &= y(t) \ . \end{aligned} \tag{6.2}$$

Vi har at numerisk approksimert løsning = eksakt løsning + feil gitt av den approksimerte løsningen. Vi kan finne feilen ved å ta den numerisk approksimerte løsningen($\hat{y}(t)$) og trekke fra den eksakte løsningen($y(t)$)

$$\hat{y}(t) - y(t) = O(\Delta t^2) \ .$$

Feilen er med andre ord lik med $O(\Delta t^2)$. Halver Δt og feilen minker med en faktor på 4, men siden vi i tillegg nå må ta dobbelt så mange steg N, gir dette at feilen minker med en faktor på 2. Det vil si at den globale feilen for implisitt Euler metoden er $O(\Delta t)$, hvilket tilsvarer en 1.ordens metode.

Stabilitet

For å vurdere stabilitet kan vi se på det enkle testproblemet

$$\begin{aligned} y' &= \lambda y \\ y(0) &= 1 \ . \end{aligned} \tag{6.3}$$

Ved å ta et tidssteg med implisitt Euler metoden på testproblemet (6.3) får vi den approksimerte løsningen

$$y(\Delta t) = \frac{1}{1 - \lambda \Delta t} = R(\lambda \Delta t) \ .$$

Vi definerer

$$R(z) = \frac{1}{1 - z} \ , \tag{6.4}$$

som er stabilitetsfunksjonen for implisitt Euler metoden. Ved å tillate at λ er et komplekst tall, så vil også z være et komplekst tall, og R(z) vil være bundet hvis vi har $|R(z)| \leq 1$. Vi kan se at siden $y(n\Delta t) = R^n(\lambda \Delta t)$, og dermed ikke eskalerer ved

store n, så vil implisitt Euler metoden være stabil når vi følger dette kravet. Vi får at stabilitetsdomenet til implisitt Euler metoden er

$$S = \{z \in C; |R(z)| \leq 1\} .$$

Med andre ord er metoden stabil for alle verdier av $z = \lambda\Delta t$ som er ligger på utsiden av enhetssirkelen med med senter (1,0), hvilket gir hele det negative halv-plan.
En numerisk metode er **A-stabil** hvis den er stabil på hele det negative halv-plan, det vil si at stabilitetsdomenet S inneholder settet

$$\{z \in C; Re(z) < 0\} .$$

L-stabilitet er et strenger krav enn A-stabilitet. En metode er L-stabil hvis den er A-stabil og

$$\lim_{z \to -\infty} R(z) = 0 .$$

Implisitt Euler metoden er både A-stabil og L-stabil.

6.1.2 Crank-Nicholson metode

Nøyaktighet

Siden det ofte kreves et veldig lite tidssteg ved 1.ordens metoder er det ønskelig å benytte en 2.ordens metode, slik som for eksempel Crank-Nicholson. For en 2.ordens metode betyr en halvering av tidssteget at feilen globalt minker med en faktor på 4. Dette vil si vår numeriske løsning vil konvergere raskere mot den eksakte løsningen enn om vi benytter oss av Euler metoden. Vi benytter samme fremgangsmåte for Crank-Nicholson metoden som for implisitt Euler, og viser hvorfor dette er en 2.ordens metode.

Vi har en differensialligning

$$\frac{dy(t)}{dt} = f(t, y(t)) \quad \in [a, b]$$

$$y(0) = y_0 .$$

Vi deler igjen intervallet [a,b] i N deler med tidssteg $\Delta t = \frac{(b-a)}{N}$ og $t_i = a + i\Delta t$ for $i = 0, 1...N$. En numerisk løsning gitt ved Crank-Nicholson metoden er

$$\begin{aligned} \frac{y_{n+1} - y_n}{\Delta t} &= \frac{1}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1})] \\ y_{n+1} - y_n &= \Delta t \frac{1}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1})] \end{aligned} \tag{6.5}$$

I Program 2 ser vi hvordan vi kan løse problemet (6.3) med Crank-Nicholson.

Program 2 Crank-Nicholson på et testproblem

```
T = 10
dt = 0.25
N = int(T/dt)
lambda1 = 0.1
y = 1
for i in range (0,N):
    y = (1+0.5*lambda1*dt)/(1-0.5*lambda1*dt)*y
print 'y = ', y
```

For å finne feilen for Crank-Nicholson metoden kan vi ta en Taylor rekkeutvikling for y rundt t_n

$$y(t) = y(t_n) + \frac{dy(t_n)}{dt}(t - t_n) + \frac{1}{2}\frac{d^2y(t_n)}{dt^2}(t - t_n)^2 + O(\Delta t^3) \, .$$

Siden

$$\frac{d^2y(t_n)}{dt^2}(t - t_n)^2 = \frac{\partial}{\partial t}[f_n] \, ,$$

og

$$\begin{aligned} \frac{\partial}{\partial t}[f_n] &= \frac{\partial f_n}{\partial t} + \frac{\partial f_n}{\partial y}\frac{\partial y}{\partial t} \\ &= \frac{\partial f_n}{\partial t} + \frac{\partial f_n}{\partial y} f_n \, , \end{aligned}$$

får vi

$$y(t) = y(t_n) + f_n(t - t_n) + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)(t - t_n)^2 + O(\Delta t^3) \, .$$

Vi evaluerer i t_{n+1} og får

$$y(t_{n+1}) = y(t_n) + \Delta t f_n + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t^2 + O(\Delta t^3)\ . \tag{6.6}$$

Vi tar en rekkeutvikling for f rundt (t_n, y_n)

$$f(t,y) = f(t_n,y_n) + \frac{\partial f(t_n,y_n)}{\partial t}(t-t_n) + \frac{\partial f(t_n,y_n)}{\partial y}(y-y_n) + O(\Delta t^2) + O(\Delta y^2)\ .$$

Evaluerer i (t_{n+1}, y_{n+1})

$$\begin{aligned} f(t_{n+1},y_{n+1}) &= f_n + \Delta t\frac{\partial f_n}{\partial t} + \Delta y\frac{\partial f_n}{\partial y} + O(\Delta t^2) + O(\Delta y^2) \\ &= f_n + \Delta t\frac{\partial f_n}{\partial t} + f_n\Delta t\frac{\partial f_n}{\partial y}\ + O(\Delta t^2) \\ f_{n+1} &= f_n + \left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t + O(\Delta t^2)\ . \end{aligned} \tag{6.7}$$

Vi setter Ligning (6.6) inn i venstre side av Ligning (6.5)

$$y_{n+1} - y_n = \Delta t f_n + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t^2 + O(\Delta t^3)\ . \tag{6.8}$$

Vi setter Ligning (6.7) inn i høyre side av Ligning (6.5)

$$\begin{aligned} \Delta t\frac{1}{2}[f(t_n,y_n) + f(t_{n+1},y_{n+1})] &= \frac{1}{2}[f_n + f_n + \left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t + O(\Delta t^2)]\Delta t \\ \Delta t\frac{1}{2}[f(t_n,y_n) + f(t_{n+1},y_{n+1})] &= \Delta t f_n + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t^2 + O(\Delta t^3)\ . \end{aligned} \tag{6.9}$$

Setter Ligning (6.8) lik med Ligning (6.9) og ser at de er like opp til $O(\triangle t^3)$. Det betyr at hvis vi halverer steget $\triangle t$, vil feilen minke med en faktor på 8. En halvering av steget $\triangle t$ betyr at vi må ta dobbelt så mange steg. Feilen minker altså med en faktor på 4 ved halvering av tidssteget. Det vil si at den globale feilen for Crank Nicholson metoden er $O(\Delta t^2)$, hvilket tilsvarer en 2.ordens metode.

Stabilitet

Vi kan finne stabilitetsfunksjonen til Crank-Nicholson metoden på lik linje som vi gjorde for implisitt Euler metoden. Vi får at testproblemet (6.3) har stabilitetsfunk-

sjonen

$$R(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \ . \tag{6.10}$$

Vi kan se at $|R(z)| \leq 1$ for Re(z)<0, og dermed er Crank-Nicholson metoden A-stabil. For L-stabilitet ser vi at

$$\lim_{z \to -\infty} R(z) = 1 \ ,$$

og metoden er dermed ikke L-stabil.

6.1.3 2.ordens Gear metode

Nøyaktighet

En annen 2.ordens metode er Gear (BDF2), som er en Backward Difference Formula(BDF). Denne metoden vises da den skal brukes i 2-stegs metoden TR-BDF2. BDF er formler som gir en aproksimasjon av den deriverte ved tiden t_n i ledd av $y(t)$ ved t_n og tidligere t. Man lager et $k'te$ grad polynom $y(t)$ ved å bruke $y(t_n),\ y(t_{n-1}),\ \ldots\ ,\ y(t_{n-k})$, derivere det og evaluere i t_n.
Vi kan bruke dette til å finne en løsning på problemet

$$\frac{dy(t)}{dt} = f(t, y(t)) \quad \in [a, b] \ .$$

Med et konstant tidssteg $\nabla t = t_n - t_{n-1}$, så kan vi ved hjelp av backward differences komme frem til et polynom av $k'te$ grad

$$y(t) = y_n + \frac{1}{\Delta t}(t - t_n)\nabla y_n + \frac{1}{2\Delta t^2}(t - t_n)(t - t_{n-1})\nabla^2 y_n + \ldots + \frac{1}{k!\Delta t^k}(t - t_n)\ldots(t - t_{n-k+1})\nabla^k y_n \ ,$$

hvor ∇ er backward difference operatoren $\nabla y_n = y_n - y_{n-1}$ og $\nabla^k y_n = \nabla^{k-1} y_n - \nabla^{k-1} y_{n-1}$. For $k = 2$ får vi en 2.ordens metode.

$$y(t) = y_n + \frac{1}{\Delta t}(t - t_n)\nabla y_n + \frac{1}{2\Delta t^2}(t - t_n)(t - t_{n-1})\nabla^2 y_n$$

$$y(t) = y_n + \frac{1}{\Delta t}(t - t_n)(y_n - y_{n-1}) + \frac{1}{2\Delta t^2}(t - t_n)(t - t_{n-1})(y_n - 2y_{n-1} + y_{n-2}) \ .$$

Vi deriverer med hensyn på t

$$y'(t) = \frac{1}{\Delta t}(y_n - y_{n-1}) + \frac{t}{\Delta t^2}(y_n - 2y_{n-1} + y_{n-2}) - \frac{1}{2\Delta t^2}(t_n + t_{n-1})(y_n - 2y_{n-1} + y_{n-2}) \ .$$

Setter $t = t_n$

$$\begin{aligned} y'(t) &= \frac{1}{\Delta t}(y_n - y_{n-1}) + \frac{1}{\Delta t^2}(y_n - 2y_{n-1} + y_{n-2})(t_n - \frac{1}{2}t_n - \frac{1}{2}t_{n-1}) \\ &= \frac{1}{\Delta t}(y_n - y_{n-1}) + \frac{1}{2\Delta t}(y_n - 2y_{n-1} + y_{n-2}) \\ &= \frac{1}{\Delta t}(\frac{3}{2}y_n - 2y_{n-1} + \frac{1}{2}y_{n-2}) \ . \end{aligned}$$

Dette gir at 2.ordens Gear metode er

$$\frac{3}{2}y_n - 2y_{n-1} + \frac{1}{2}y_{n-2} = \Delta t f(y_n, t_n) \ .$$

Vi ganger alle ledd med $\frac{2}{3}$ og får

$$y_n - \frac{4}{3}y_{n-1} + \frac{1}{3}y_{n-2} = \frac{2}{3}\Delta t f(y_n, t_n) \ .$$

Vi ønsker å utrykke metoden ved hjelp av y_{n+1}, hvilket gir

$$y_{n+1} - \frac{4}{3}y_n + \frac{1}{3}y_{n-1} = \frac{2}{3}\Delta t f(y_{n+1}, t_{n+1}) \ . \tag{6.11}$$

Ved hjelp av Taylor rekkeutvikling kan vi se at dette faktisk er en 2.ordens metode. Vi starter med en rekkeutvikling for y rundt t_n

$$\begin{aligned} y(t) &= y(t_n) + \frac{dy(t_n)}{dt}(t - t_n) + \frac{1}{2}\frac{d^2y(t_n)}{dt^2}(t - t_n)^2 + O(\Delta t^3) \\ &= y(t_n) + f_n(t - t_n) + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)(t - t_n)^2 + O(\Delta t^3) \ . \end{aligned}$$

Vi evaluerer i t_{n+1} og t_{n-1}

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \Delta t f_n + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t^2 + O(\Delta t^3) \\ y(t_{n-1}) &= y(t_n) - \Delta t f_n + \frac{1}{2}\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\Delta t^2 + O(\Delta t^3) \ . \end{aligned}$$

Vi setter dette inn i venstre side av Ligning (6.11)

$$\begin{aligned} v.s &= y_n + \Delta t f_n + \frac{1}{2}\Delta t^2\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right) - \frac{4}{3}y_n \\ &\quad + \frac{1}{3}\left(y_n - \Delta t f_n + \frac{1}{2}\Delta t^2\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right)\right) + O(\Delta t^3) \\ &= \frac{2}{3}\Delta t f_n + \frac{2}{3}\Delta t^2\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right) + O(\Delta t^3) \ . \end{aligned}$$

Deretter tar vi en rekkeutvikling for f rundt (t_n, y_n), og evaluerer i (t_{n+1}, y_{n+1}). Dette gir

$$f_{n+1} = f_n + \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) \Delta t + O(\Delta t^2) \ .$$

Vi setter dette inn i høyre side av Ligning (6.11)

$$\begin{aligned} h.s &= \frac{2}{3}\Delta t \left(f_n + \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) \Delta t + O(\Delta t^2) \right) \\ &= \frac{2}{3}\Delta t f_n + \frac{2}{3}\Delta t^2 \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) + O(\Delta t^3) \ . \end{aligned}$$

Dermed ser vi at venstre side og høyre side av Ligning (6.11) er like opp til $O(\triangle t^3)$. Det vil si at den globale feilen for Gear metoden er $O(\Delta t^2)$, hvilket tilsvarer en 2.ordens metode.

6.1.4 TR-BDF2

Nøyaktighet

TR-BDF2 er en type Composite-Backward Difference Formula (C-BDF). Dette er en 1-stegs 2-nivåers metode, hvilket vil si at den har to indre steg, som er sammensatt av trapesmetoden og BDF2 [21]. Vi har en differensialligning

$$\frac{dy(t)}{dt} = f(t, y(t)) \quad \in [a, b]$$

$$y(0) = y_0 \ .$$

TR-BDF2 består av to steg hvor vi først bringer løsningen vår fra t_n til $t_{n+2\gamma}$ ved å bruke 2.ordens trapesmetoden.

$$y_{n+2\gamma} - \gamma \Delta t f(y_{n+2\gamma}) = y_n + \gamma \Delta t f(y_n) \ . \tag{6.12}$$

Det andre steget bringer løsningen fra $t_{n+2\gamma}$ til t_{n+1} ved hjelp av 2.ordens metoden BDF2

$$y_{n+1} - \gamma_2 \Delta t f(y_{n+1}) = \frac{1-\gamma_2}{2\gamma} y_{n+2\gamma} + \left(1 - \frac{1-\gamma_2}{2\gamma} \right) y_n \ , \tag{6.13}$$

hvor

$$\gamma_2 = \frac{1-2\gamma}{2(1-\gamma)} \,. \tag{6.14}$$

TR-BDF2 er av 2.ordens nøyaktighet hvis $\gamma \neq 0, 1$. Ved den spesielle verdien $\gamma = 1/2$ vil TR-BDF2 skjemaet falle sammen med trapesmetoden [21]. Vi velger $\gamma = 1 - \frac{\sqrt{2}}{2}$, hvilket sørger for stabilitet av metoden.

Program 3 viser hvordan vi kan løse problemet (6.3) med TR-BDF2.

Program 3 TR-BDF2 på et testproblem

```
T = 10
dt = 0.25
N = int(T/dt)
lambda1 = 0.1
gamma = 1 - math.sqrt(2)/2.0
gamma2 = (1-(2*gamma))/(2*(1-gamma))
y = 1
for i in range (0,N):
    y_mid = (1+gamma*lambda1*dt)/(1-gamma*lambda1*dt)*y
    y = ((1-gamma2)/(2*gamma)*y_mid + (1-(1-gamma2)
        /(2*gamma))*y)/(1-(gamma2*lambda1*dt))
print 'y = ', y
```

Ved hjelp av Taylor rekkeutvikling kan vi se at dette faktisk er en 2.ordens metode. Vi ser først på 1. (indre) steg, Ligning (6.12), i metoden. Omskriver til

$$y_{n+2\gamma} - y_n = \gamma\Delta t(f(y_{n+2\gamma}) + f(y_n)) \,. \tag{6.15}$$

Vi tar en rekkeutvikling for y rundt t_n, og evaluerer i $t_{n+2\gamma}$

$$y_{n+2\gamma} = y_n + \Delta t 2\gamma f_n + \Delta t^2\gamma^2 2\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f_n}{\partial y}\right) + O(\Delta t^3) \,.$$

Deretter tar vi en rekkeutvikling for f rundt (t_n, y_n), og evaluerer i $(t_{n+2\gamma}, y_{n+2\gamma})$

$$f_{n+2\gamma} = f_n + 2\gamma\left(\frac{\partial f_n}{\partial t} + f_n\frac{\partial f}{\partial y}\right)\Delta t + O(\Delta t^2) \,.$$

Vi setter inn for $y_{n+2\gamma}$ og $f_{n+2\gamma}$ på venstre og høyre side av Ligning (6.15)

$$\begin{aligned} v.s. &= y_{n+2\gamma} - y_n \\ &= \Delta t 2\gamma f_n + \Delta t^2 \gamma^2 2 \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) + O(\Delta t^3) \ . \\ h.s &= \gamma \Delta t \left(f(y_{n+2\gamma}) + f(y_n) \right) \\ &= \gamma \Delta t \left(f_n + 2\gamma \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f}{\partial y} \right) \Delta t + O(\Delta t^2) + f_n \right) \\ &= \Delta t 2\gamma f_n + \Delta t^2 \gamma^2 2 \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) + O(\Delta t^3) \ . \end{aligned}$$

Vi ser at venstre side og høyre side er like opp til $O(\triangle t^3)$. Det vil si at den globale feilen for 1. (indre) steg av TR-BDF2 metoden er $O(\Delta t^2)$.

Vi ser deretter på 2. (indre) steg, Ligning (6.13), i metoden. Omskriver til

$$y_{n+1} - \frac{1-\gamma_2}{2\gamma} y_{n+2\gamma} - \left(1 - \frac{1-\gamma_2}{2\gamma} \right) y_n = \gamma_2 \Delta t f(y_{n+1}) \ . \tag{6.16}$$

Vi har at

$$\frac{1-\gamma_2}{2\gamma} = \frac{1}{4\gamma(1-\gamma)} \ .$$

Vi tar en rekkeutvikling for y rundt t_n, og evaluerer i t_{n+1}

$$y_{n+1} = y_n + \Delta t f_n + \frac{\Delta t^2}{2} \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) + O(\Delta t^3) \ .$$

Deretter tar vi en rekkeutvikling for f rundt (t_n, y_n), og evaluerer i (t_{n+1}, y_{n+1})

$$f_{n+1} = f_n + \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y} \right) \Delta t + O(\Delta t^2) \ .$$

Vi setter inn for y_{n+1} og $y_{n+2\gamma}$ på venstre side av Ligning (6.16)

$$\begin{aligned}
v.s &= y_{n+1} - \frac{1-\gamma_2}{2\gamma} y_{n+2\gamma} - \left(1 - \frac{1-\gamma_2}{2\gamma}\right) y_n \\
&= y_n + \Delta t f_n + \frac{\Delta t^2}{2}\left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) \\
&- \frac{1-\gamma_2}{2\gamma}\left(y_n + \Delta t 2\gamma f_n + \Delta t^2 \gamma^2 2\left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right)\right) - \left(1 - \frac{1-\gamma_2}{2\gamma}\right) y_n + O(\Delta t^3) \\
&= \Delta t f_n - \frac{\Delta t}{2(1-\gamma)} f_n + \frac{\Delta t^2}{2}\left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) - \frac{\Delta t^2}{2(1-\gamma)}\left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) + O(\Delta t^3) \\
&= \frac{1-2\gamma}{2(1-\gamma)} \Delta t f_n + \frac{1-2\gamma}{2(1-\gamma)} \Delta t^2 \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) + (\Delta t^3) \ .
\end{aligned}$$

Setter inn for f_{n+1} på høyre side av Ligning (6.16)

$$\begin{aligned}
h.s &= \gamma_2 \Delta t f_{n+1} \\
&= \frac{1-2\gamma}{2(1-\gamma)} \Delta t \left(f_n + \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) \Delta t + O(\Delta t^2)\right) \\
&= \frac{1-2\gamma}{2(1-\gamma)} \Delta t f_n + \frac{1-2\gamma}{2(1-\gamma)} \Delta t^2 \left(\frac{\partial f_n}{\partial t} + f_n \frac{\partial f_n}{\partial y}\right) + O(\Delta t^3) \ .
\end{aligned}$$

Vi ser at venstre side og høyre side er like opp til $O(\triangle t^3)$. Det vil si at den globale feilen for 2. (indre) steg av TR-BDF2 metoden er $O(\Delta t^2)$. Dette betyr at TR-BDF2 er en 2.ordens metode.

Stabilitet

Vi kan finne stabilitetsfunksjonen til TR-BDF2 metoden på lik linje som vi gjorde for implisitt Euler metoden. For et testproblem (6.3) får vi at

$$R(z) = \frac{(1-\gamma_2)(1+\gamma z) - (2\gamma - 1 - \gamma_2)(1-\gamma z)}{2\gamma(1-\gamma z)(1-\gamma_2 z)} \ . \tag{6.17}$$

Siden $|R(z)| \leq 1$ for Re(z)<0, så er TR-BDF2 A-stabil. I tillegg er

$$\lim_{z \to -\infty} R(z) = 0 \ ,$$

slik at metoden også er L-stabil.

6.2 Operator splitting

Operator splitting er en attraktiv teknikk for å løse koblede systemer av PDE'er. Komplekse ligningssystemer kan splittes i mindre deler slik at de blir lettere å løse [16]. Vi skal se nærmere på to måter å splitte et ligningssystem på; Godunov og Strang splitting.

6.2.1 Godunov splitting

Vi har et enkelt problem på formen

$$\frac{du}{dt} = (L_1 + L_2)u \quad t \in [0, b] \tag{6.18}$$

$$u(0) = u_0$$

hvor L_1 og L_2 er operatorer på u som ikke avhenger eksplisitt av t, og u_0 er startverdi. Vi kan velge et lite tidssteg $\triangle$t, og kan dermed finne en approksimert løsning ved t = Δt ved å løse problemet

$$\frac{dv}{dt} = L_1(v) \quad t \in [0, \Delta t]$$

$$v(0) = u_0 \ ,$$

for deretter løse problemet

$$\frac{dw}{dt} = L_2(w) \quad t \in [0, \Delta t]$$

$$w(0) = v(\Delta t) \ .$$

Program 4 Godunov splitting

```
for i in xrange(0, N):
    t = i*dt
    ode.forward(v, t, dt)
    pde.solve(v)
```

Vi kan i Program 4 se hvoran vi kan bruke Godunov splitting på Monodomene-modellen. For hvert tidssteg t løser $ode.forward(v, t, dt)$ ODE problemet fra t til $t + dt$ og $pde.solve(v)$ løser PDE problemet fra t til $t + dt$.

Vi har at $w(\Delta t)$ er en approksimasjon til $u(\Delta t)$, og dette kan vi bevise gjennom en

Taylor rekkeutvikling.Vi starter med en rekkeutvikling av u rundt t_0 og evaluerer i $t_{0+\Delta t}$

$$u(\Delta t) = u_0 + \Delta t \frac{du_0}{dt} + \frac{\Delta t^2}{2}\frac{d^2u_0}{dt^2} + O(\Delta t^3) \ .$$

Fra Ligning (6.18) har vi

$$\frac{du}{dt} = (L_1 + L_2)u \ .$$

Mens den andre deriverte blir

$$\frac{d^2u}{dt^2} = (L_1 + L_2)^2 u \ .$$

Vi setter inn i Taylor rekken

$$u(\Delta t) = u_0 + \Delta t(L_1 + L_2)u_0 + \frac{\Delta t^2}{2}(L_1 + L_2)^2 u_0 + O(\Delta t^3) \ . \tag{6.19}$$

Rekkeutvikler på samme måte for $v(\Delta t)$, og for $w(\Delta t)$ hvor $v(\Delta t)$ er startverdi

$$\begin{aligned} v(\Delta t) &= v_0 + \Delta t \frac{dv_0}{dt} + \frac{\Delta t^2}{2}\frac{d^2v_0}{dt^2} + O(\Delta t^3) \\ &= u_0 + \Delta t L_1 u_0 + \frac{\Delta t^2}{2} L_1^2 u_0 + O(\Delta t^3) \ . \end{aligned}$$

$$\begin{aligned} w(\Delta t) &= w_0 + \Delta t \frac{dw_0}{dt} + \frac{\Delta t^2}{2}\frac{d^2w_0}{dt^2} + O(\Delta t^3) \\ &= v(\Delta t) + \Delta t L_2 v(\Delta t) + \frac{\Delta t^2}{2} L_2^2 v(\Delta t) + O(\Delta t^3) \ . \end{aligned}$$

Setter inn for $v(\Delta t)$ i $w(\Delta t)$ og får

$$w(\Delta t) = u_0 + \Delta t(L_1 + L_2)u_0 + \frac{\Delta t^2}{2}(L_1^2 + 2L_1L_2 + L_2^2)u_0 + O(\Delta t^3) \ . \tag{6.20}$$

Feilen ved $t = \Delta t$ er gitt ved forskjellen mellom Ligning (6.19) og Ligning (6.20)

$$w(\Delta t) - u(\Delta t) = \frac{\Delta t^2}{2}(L_2L_1 - L_1L_2)u_0 + O(\Delta t^3) \ .$$

Feilen etter et tidssteg Δt er proporsjonal med Δt^2, og den totale feilen er $n\Delta t^2$ etter n tidssteg. Antall steg n er proporsjonal med Δt^{-1}, og feilen ved $t = b$ er dermed proporsjonal med Δt. Godunov splitting er med andre ord en 1.ordens splitte metode.

6.2.2 Strang splitting

Å løse undersystemene med en høyere orden enn orden for splitte metoden har ingen hensikt, da nøyaktigheten vil være begrenset av splittefeilen. For å kunne å få utbytte av undersystemenes 2.ordens nøyaktighet må vi ha en 2.ordens splittemetode. Dette kan løses ved å bruke Strang [5] splitting. Problemet (6.18) kan da løses på følgende måte

$$v_t = L_1(v) \quad t \in [0, \Delta t/2]$$

$$v(0) = u_0 \ ,$$

deretter

$$w_t = L_2(w) \quad t \in [0, \Delta t]$$

$$w(0) = v\left(\Delta t/2\right) \ ,$$

og til slutt

$$v_t = L_1(v) \quad t \in [\Delta t/2, \Delta t]$$

$$v\left(\Delta t/2\right) = w(\Delta t) \ .$$

Vi kan i Program 5 se hvoran vi kan bruke Strang splitting på Monodomenemodellen. For hvert tidssteg t løser $ode.forward(v, t, 0.5{*}dt)$ ODE problemet fra t til $t{+}\frac{1}{2}\Delta t$, $pde.solve(v)$ løser PDE problemet fra t til $t+\Delta t$ og $ode.forward(v, t+0.5{*}dt, 0.5{*}dt)$ løser ODE problemet fra $t + \frac{1}{2}\Delta t$ til $t + \Delta t$.

Program 5 Strang splitting

```
for i in xrange(0, N):
    t = i*dt
    ode.forward(v, t, 0.5*dt)
    pde.solve(v)
    ode.forward(v, t + 0.5*dt, 0.5*dt)
```

Vi kan vise at Strang splitting er en 2.ordens metode ved å ta en rekkeutvikling av løsningen v ved $t = \triangle t$. Vi startet først med en rekkeutvikling av v rundt t_0 og evaluerer i $t_{0+\Delta t/2}$

$$\begin{aligned} v(\Delta t/2) &= v_0 + \frac{\Delta t}{2}\frac{dv_0}{dt} + \frac{\Delta t^2}{8}\frac{d^2v_0}{t^2} + O(\Delta t^3) \\ &= u_0 + \frac{\Delta t}{2}L1u_0 + \frac{\Delta t^2}{8}L_1^2u_0 + O(\Delta t^3) \ . \end{aligned}$$

Deretter en rekkeutvikling av $w(\Delta t)$ med $v(\Delta t/2)$ som startverdi

$$\begin{aligned} w(\Delta t) &= w_0 + \Delta t \frac{dw_0}{dt} + \frac{\Delta t^2}{2}\frac{d^2w}{dt^2} + O(\Delta t^3) \\ &= u_0 + (\frac{\Delta t}{2}L_1 + \Delta t L_2)u_0 + \frac{\Delta t^2}{8}L_1^2 u_0 + \frac{\Delta t^2}{2}L_2 L_1 u_0 + \frac{\Delta t^2}{2}L_2^2 u_0 + O(\Delta t^3)\ . \end{aligned}$$

Tilslutt en rekkeutvikling for $v(\Delta t)$ med $w(\Delta t)$ som startverdi

$$\begin{aligned} v(\Delta t) &= v_0 + \frac{\Delta t}{2}\frac{dv_0}{dt} + \frac{\Delta t^2}{8}\frac{d^2v}{dt^2} + O(\Delta t^3) \\ &= v_0 + \frac{\Delta t}{2}L_1 v_0 + \frac{\Delta t^2}{8}L_1^2 v_0 + O(\Delta t^3) \\ &= u_0 + \left(\frac{\Delta t}{2}L_1 + \Delta t L_2\right)u_0 + \left(\frac{\Delta t^2}{8}L_1^2 + \frac{\Delta t^2}{2}L_2 L_1 + \frac{\Delta t^2}{2}L_2^2\right)u_0 \\ &+ \frac{\Delta t}{2}L_1\left(u_0 + \frac{\Delta t}{2}L_1 u_0 + \Delta t L_2 u_0\right) + \frac{\Delta t^2}{8}L_1^2 u_0 + O(\Delta t^3)\ . \end{aligned}$$

Forenkler $v(\Delta t)$

$$\begin{aligned} v(\Delta t) &= u_0 + \Delta t(L_1 + L_2)u_0 + \frac{\Delta t^2}{2}(L_1^2 + L_1 L_2 + L_2 L_1 + L_2^2)u_0 + O(\Delta t^3) \\ &= u_0 + \Delta t(L_1 + L_2)u_0 + \frac{\Delta t^2}{2}(L_1 + L_2)^2 u_0 + O(\Delta t^3)\ . \end{aligned} \tag{6.21}$$

Feilen ved $t = \Delta t$ er gitt ved forskjellen mellom ligning (6.19) og ligning (6.21)

$$v(\Delta t) - u(\Delta t) = O(\Delta t^3)\ .$$

Feilen er proporsjonal med $\triangle t^3$ etter et tidssteg, mens feilen etter n ~Δt^{-1} tidssteg er feilen proporsjonal med Δt^2. Strang splitting er med andre ord en 2.ordens splittemetode.

6.3 Diskretisering av modellene

For å kunne løse Bidomene- og Monodomenemodellen numerisk er vi avhengig av å diskretisere modellene.

6.3.1 Diskretisering av Bidomenemodellen

Det mest stabile er å kunne løse systemet på en fullt implisitt måte, men på grunn av at de innvolverte ODE'ene er kompliserte så er dette vanskelig. Derfor er det

vanlig å bruke operator splitting for å forenkle systemet vårt.
Vi kan skrive System (5.1) som

$$\begin{aligned}
\frac{\partial s}{\partial t} &= F(s,V,t) && x \in H \qquad (6.22)\\
\alpha\frac{\partial V}{\partial t} + G(V) &= \nabla\cdot(M_i\nabla V) + \nabla\cdot(M_i\nabla u_e) && x \in H\\
\nabla\cdot(M_i\nabla V) &= -\nabla\cdot((M_e+M_i)\nabla u_e) && x \in H\\
V(x,0) &= I && x \in H\\
n\cdot(M_i\nabla V + M_i\nabla u_e) &= 0 && x \in \partial H\\
n\cdot(M_e\nabla u_e) &= 0 && x \in \partial H
\end{aligned}$$

hvor $\alpha = \chi C_m$ og $G(V) = \chi I_{ion} - \chi I_{app}$.

Systemet splittes i to deler.

$$\begin{aligned}
\frac{\partial V}{\partial t} &= -G(V)\\
\frac{\partial s}{\partial t} &= F(s,V,t)
\end{aligned}$$

og

$$\begin{aligned}
\alpha\frac{\partial V}{\partial t} &= \nabla\cdot(M_i\nabla V) + \nabla\cdot(M_i\nabla u_e)\\
\nabla\cdot(M_i\nabla V) &= -\nabla\cdot((M_e+M_i)\nabla u_e)
\end{aligned}$$

Vi ønsker å løse systemet ved hjelp av Strang splitting slik at vi får 2.ordens nøyaktighet.Vi antar at $V^n = V(t_n)$ og $s^n = s(t_n)$ er kjent.

Metoden er som følger:

1. Løs systemet

$$\begin{aligned}
\frac{\partial V}{\partial t} &= -G(V) && V(t_n) = V^n\\
\frac{\partial s}{\partial t} &= F(s,V,t) && s(t_n) = s^n
\end{aligned}$$

for $t_n < t < t_n + \frac{1}{2}\Delta t$. Dette gir de approksimerte løsningene $s^* = s(t_n + \frac{1}{2}\Delta t)$ og $V^* = V(t_n + \frac{1}{2}\Delta t)$.

2. Løs den lineære PDE-systemet

$$\alpha \frac{\partial V}{\partial t} = \nabla \cdot (M_i \nabla V) + \nabla \cdot (M_i \nabla u_e) \tag{6.23}$$

$$\nabla \cdot (M_i \nabla V) = -\nabla \cdot ((M_e + M_i) \nabla u_e) \tag{6.24}$$

med $V(t_n) = V^*$ for $t_n < t < t_n + \Delta t$. Dette gir den approksimerte løsningen $V^{**} = V(t_n + \Delta t)$.

3. Løs systemet

$$\frac{\partial V}{\partial t} = -G(V) \qquad V\left(t_n + \frac{1}{2}\Delta t\right) = V^{**}$$

$$\frac{\partial s}{\partial t} = F(s, V, t) \qquad s\left(t_n + \frac{1}{2}\Delta t\right) = s^*$$

for $t_n + \frac{1}{2}\Delta t < t < t_n + \Delta t$. Dette gir den approksimerte løsningen V^{n+1} og s^{n+1} ved $t = t_n + \Delta t$

Svak variasjonsformulering av PDE-problemet

Vi ønsker å komme frem til en svak variasjonsformulering for de lineære PDE'ene. Dette gjøres ved å multiplisere med en testfunksjon $w \in \mathbf{V}$ hvor $\mathbf{V}$ er et funksjonsrom , og integrer over Ω. For Ligning (6.23) får vi

$$\alpha \int_\Omega \frac{\partial V}{\partial t} w \, d\Omega = \int_\Omega \left(\nabla \cdot (M_i \nabla V) + \nabla \cdot (M_i \nabla u_e)\right) w \, d\Omega \, . \tag{6.25}$$

Fra Green's teorem har vi

$$\int_\Omega \left(\nabla \cdot (M_i \nabla V) + \nabla \cdot (M_i \nabla u_e)\right) w \, d\Omega = -\int_\Omega (M_i \nabla V + M_i \nabla u_e) \nabla w \, d\Omega + \int_{d\Omega} n \cdot (M_i \nabla V + M_i \nabla u_e) \, d\Gamma \, . \tag{6.26}$$

Siden vi har randbetingelsen $n \cdot (M_i \nabla V + M_i \nabla u_e) = 0$ på $d\Omega$ blir Ligning (6.26)

$$\int_\Omega \nabla \cdot (M_i \nabla V) + \nabla \cdot (M_i \nabla u_e) w \, d\Omega = -\int_\Omega (M_i \nabla V + M_i \nabla u_e) \nabla w \, d\Omega \, . \tag{6.27}$$

Dermed er randbetingelsen implisitt uttrykt i ligningen ved hjelp av Green's teorem. Vi kombinerer nå Ligning (6.25) med Ligning (6.27)

$$\alpha \int_\Omega \frac{\partial V}{\partial t} w \, d\Omega = - \int_\Omega (M_i \nabla V + M_i \nabla u_e) \nabla w \, d\Omega \, . \tag{6.28}$$

For Ligning (6.24) får vi

$$\int_\Omega \nabla \cdot (M_i \nabla V) w \, d\Omega + \int_\Omega \nabla \cdot ((M_e + M_i) \nabla u_e) w \, d\Omega = 0 \, . \tag{6.29}$$

Ved hjelp av Green's teorem og randbetingelsene får vi

$$\int_\Omega M_i \nabla V \nabla w \, d\Omega + \int_\Omega (M_e + M_i) \nabla u_e \nabla w \, d\Omega = 0 \, . \tag{6.30}$$

Definerer

$$\begin{aligned} (v, w) &= \int_\Omega vw \, d\Omega && v, w \in \mathbf{V} \\ a_I(v, w) &= \int_\Omega M_i \nabla v \nabla w \, d\Omega \\ a_{I+E}(v, w) &= \int_\Omega (M_i + M_e) \nabla v \nabla w \, d\Omega \, . \end{aligned}$$

Dermed kan vi skrive en svak variasjonsformulering for PDE-problemet:

Finn $V, u_e \in \mathbf{V}$ slik at for alle $w \in \mathbf{V}$

$$\begin{aligned} \alpha \left(\frac{\partial V(x,t)}{\partial t}, w \right) &= -a_I(V(x,t), w) - a_I(u_e, w) \\ a_I(V(x,t), w) + a_{I+E}(u_e, w) &= 0 \\ V(x, 0) &= I \, . \end{aligned} \tag{6.31}$$

Diskretisering i tid

Diskretisering i tid kan gjøres ved hjelp av θ-regelen.Ligningen

$$\frac{du}{dt} = G$$

approksimeres med

$$\frac{v^{n+1} - v^n}{\Delta t} = \theta G^{n+1} + (1 - \theta) G^n \qquad 0 \leq \theta \leq 1$$

Ved å velge $\theta = 0$ får vi et forlengs (eksplisitt) Euler skjema, $\theta = 1$ gir et implisitt Euler skjema, mens $\theta = 1/2$ gir Crank-Nicholson skjema.
Bruker θ-regelen på System (6.31)

$$\begin{aligned}\alpha \int_\Omega \frac{v^{n+1} - v^n}{\Delta t} w \, d\Omega &= -\theta \int_\Omega (M_i \nabla v^{n+1} + M_i \nabla u_e^{n+\theta}) \nabla w \, d\Omega \\ &\quad - (1-\theta) \int_\Omega (M_i \nabla v^n + M_i \nabla u_e^{n+\theta}) \nabla w \, d\Omega \\ \theta \int_\Omega M_i \nabla v^{n+1} \nabla w \, d\Omega + \int_\Omega (M_i + M_e) \nabla u_e^{n+\theta} \nabla w \, d\Omega &= -(1-\theta) \int_\Omega M_i \nabla v^n \nabla w \, d\Omega \, ,\end{aligned}$$

hvor $u_e^{n+\theta}$ approksimerer u_e ved tidssteg $t_n + \theta \Delta t$.

Diskretisering i rommet

Introduserer et endelig dimensjonalt underrom $\mathbf{V_h} \in \mathbf{V}$ med stykkvise lineære basisfunksjoner $\{\phi_1(x), ..., \phi_n(x)\}$ som danner basisen for $\mathbf{V_h}$. Approksimerer v^{n+1} og $u_e^{n+\theta}$ som en lineær kombinasjon av basisfunksjonene.

$$\begin{aligned} v^{n+1} &\approx v_h = \sum_{j=1}^{n} v_j \phi_j \\ u_e^{n+\theta} &\approx u_h = \sum_{j=1}^{n} u_j \phi j \, . \end{aligned}$$

Ved bruk av Galerkins metode benyttes de samme basisfunksjonene for å approksimere vektfunksjonene, $w = \phi_i$ for $i = 1, ..., n$. Det diskre problemet blir

Finn $v_h, u_h \in \mathbf{V_h}$ slik at for alle $\phi_i \in \mathbf{V_h}$

$$\begin{aligned}\alpha \int_\Omega v_h \phi_i \, d\Omega + \Delta t \theta \int_\Omega M_i \nabla v_h \nabla \phi_i \, d\Omega + \Delta t \int_\Omega M_i \nabla u_h \nabla \phi_i \, d\Omega \\ = \alpha \int_\Omega v^n \phi_i \, d\Omega - (1-\theta) \Delta t \int_\Omega M_i \nabla v^n \nabla \phi_i \, d\Omega\end{aligned} \tag{6.32}$$

$$\theta \int_\Omega M_i \nabla v_h \nabla \phi_i \, d\Omega + \int_\Omega (M_i + M_e) \nabla u_h \nabla \phi_i \, d\Omega = -(1-\theta) \int_\Omega M_i \nabla v^n \nabla \phi_i \, d\Omega \, . \tag{6.33}$$

Element formulering

Setter inn for v_h og u_h. For Ligning (6.32) får vi

$$\sum_{j=1}^{n} v_j \alpha \int_\Omega \phi_j \phi_i \, d\Omega + \theta \Delta t \sum_{j=1}^{n} v_j \int_\Omega M_i \nabla \phi_j \nabla \phi_i \, d\Omega + \Delta t \sum_{j=1}^{n} u_j \int_\Omega M_i \nabla \phi_j \nabla \phi_i \, d\Omega$$
$$= \alpha \int_\Omega v^n \phi_i \, d\Omega - (1-\theta)\Delta t \int_\Omega M_i \nabla v^n \nabla \phi_i \, d\Omega \qquad i = 1, .., n$$

Her bruker vi definisjon av indreproduktene $(\ ,\ )$:

$$\alpha \sum_{j=1}^{n} v_j (\phi_j \phi_i) + \theta \Delta t \sum_{j=1}^{n} v_j a_I(\phi_j, \phi_i) + \Delta t \sum_{j=1}^{n} u_j a_I(\phi_j, \phi_i)$$
$$= \alpha (v^n, \phi_i) - \Delta t (1-\theta) a_I(v^n, \phi_i) \qquad i = 1, .., n$$

For Ligning (6.33) multipliserer vi først med $\Delta t/\theta$

$$\Delta t \sum_{j=1}^{n} v_j \int_\Omega M_i \nabla \phi_j \nabla \phi_i \, d\Omega + \frac{\Delta t}{\theta} \sum_{j=1}^{n} u_j \int_\Omega (M_i + M_e) \nabla \phi_j \nabla \phi_i \, d\Omega$$
$$= -\Delta t \frac{(1-\theta)}{\theta} \int_\Omega M_i \nabla v^n \nabla \phi_i \, d\Omega \qquad i = 1, .., n$$

og deretter bruker vi definisjon av indreproduktene $(\ ,\ )$:

$$\Delta t \sum_{j=1}^{n} v_j a_I(\phi_j, \phi_i) + \frac{\Delta t}{\theta} \sum_{j=1}^{n} u_j a_{I+E}(\phi_j, \phi_i) = -\Delta t \frac{(1-\theta)}{\theta} a_I(v^n, \phi_i) \qquad i = 1, .., n$$

Vi får et lineært system av ligninger med de ukjente u_j, v_j for $j = 1, .., n$. Det lineære systemet kan skrives som

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

hvor

$$A_{ij} = (\phi_j, \phi_i) + \theta \Delta t a_I(\phi_j, \phi_i)$$
$$B_{ij} = \Delta t a_I(\phi_j, \phi_i)$$
$$C_{ij} = \frac{\Delta t}{\theta} a_{I+E}(\phi_j, \phi_i)$$
$$a_i = (v^n, \phi_i) - \Delta t (1-\theta) a_I(v^n, \phi_i)$$
$$b_i = -\Delta t \frac{1-\theta}{\theta} a_I(v^n, \phi_i)$$

6.3.2 Diskretisering av Monodomenemodellen

Vi kan skrive System (5.17) som

$$\begin{aligned} \frac{\partial s}{\partial t} &= F(s,V,t) && x \in H \\ \alpha\frac{\partial V}{\partial t} + G(V) &= \nabla \cdot M\nabla V && x \in H \\ V(x,0) &= I && x \in H \\ n^T M\nabla V &= 0 && x \in \partial H \end{aligned} \tag{6.34}$$

hvor $\alpha = \chi C_m$ og $G(V) = \chi I_{ion} - \chi I_{app}$.

På samme måte som for Bidomenemodellen, så splittes systemet i to deler.

$$\begin{aligned} \frac{\partial V}{\partial t} &= -G(V) \\ \frac{\partial s}{\partial t} &= F(s,V,t) \end{aligned}$$

og

$$\alpha\frac{\partial V}{\partial t} = \nabla \cdot M\nabla V \ .$$

Vi ønsker nok en gang å løse systemet ved hjelp av Strang splitting slik at vi får 2.ordens nøyaktighet. Vi antar at $V^n = V(t_n)$ og $s^n = s(t_n)$ er kjent.
Metoden er som følger:

1. Løs systemet

$$\begin{aligned} \frac{\partial V}{\partial t} &= -G(V) && V(t_n) = V^n \\ \frac{\partial s}{\partial t} &= F(s,V,t) && s(t_n) = s^n \end{aligned}$$

   for $t_n < t < t_n + \frac{1}{2}\Delta t$. Løsningen V og s ved $t = t_n + 1/2\Delta t$ angis som hhv V^* og s^*.

2. Løs den lineære PDE'en

$$\alpha\frac{\partial V}{\partial t} = \nabla \cdot M\nabla V \qquad V(t_n) = V^*$$

   for $t_n < t < t_n + \Delta t$. Løsningen V ved $t = t_n + \Delta t$ angis som V^{**}

3. Løs systemet

$$\frac{\partial V}{\partial t} = -G(V) \qquad V\left(t_n + \frac{1}{2}\Delta t\right) = V^{**}$$
$$\frac{\partial s}{\partial t} = F(s, V, t) \qquad s\left(t_n + \frac{1}{2}\Delta t\right) = s^{*}$$

for $t_n + \frac{1}{2}\Delta t < t < t_n + \Delta t$. Dette gir den approksimerte løsningen V^{n+1} og s^{n+1} ved $t = t_n + \Delta t$

Svak variasjonsformulering av PDE-problemet

Vi starter med å komme frem til en svak variasjonsformulering for den lineære PDE'en. Dette gjøres ved å multiplisere med en testfunksjon $w \in \mathbf{V}$ hvor $\mathbf{V}$ er et funksjonsrom, og integrerer over Ω

$$\alpha \int_\Omega \frac{\partial V}{\partial t} w \, d\Omega = \int_\Omega \nabla \cdot (M \nabla V) w \, d\Omega \, . \tag{6.35}$$

Fra Green's teorem har vi

$$\int_\Omega (\nabla \cdot M \nabla V) w \, d\Omega = -\int_\Omega M \nabla V \nabla w \, d\Omega + \int_{d\Omega} n \cdot M \nabla V w \, d\Gamma \, . \tag{6.36}$$

Siden vi har randbetingelsen $n \cdot M \nabla V = 0$ på $d\Omega$ blir Ligning (6.36)

$$\int_\Omega \nabla \cdot M \nabla V w \, d\Omega = -\int_\Omega M \nabla V \nabla w \, d\Omega \tag{6.37}$$

Dermed er randbetingelsen implisitt uttrykt i ligningen ved hjelp av Green's teorem. Vi kombinerer nå Ligning (6.35) med Ligning (6.37)

$$\alpha \int_\Omega \frac{\partial V}{\partial t} w \, d\Omega = -\int_\Omega M \nabla V \nabla w \, d\Omega \, . \tag{6.38}$$

Vi kan definere to indreprodukt $(\ ,\ )$:

$$(v, w) = \int_\Omega vw \, d\Omega \qquad v, w \in \mathbf{V}$$
$$a(v, w) = \int_\Omega M \nabla v \nabla w \, d\Omega \, .$$

Dermed kan vi skrive en svak variasjonsformulering for PDE-problemet:

Finn en funksjon $V \in \mathbf{V}$ slik at for alle $w \in \mathbf{V}$

$$\alpha \left(\frac{\partial V(x,t)}{\partial t}, w \right) = -a(V(x,t), w)$$
$$V(x,0) = I \ .$$

Diskretisering i tid

Som for Bidomenemodellen kan vi diskretisere i tid ved hjelp av θ-regelen. Bruker θ-regelen på Ligning (6.38)

$$\alpha \int_\Omega \frac{v^{n+1} - v^n}{\Delta t} w \ d\Omega = -\int_\Omega \theta M \nabla v^{n+1} \nabla w \ d\Omega - (1-\theta) \int_\Omega M \nabla v^n \nabla w \ d\Omega \quad i = 1, .., n \tag{6.39}$$

Diskretisering i rommet

På samme måte som tidligere introduserer vi et endelig dimensjonalt underrom $\mathbf{V_h} \in \mathbf{V}$ med stykkvise lineære basisfunksjoner $\{\phi_1(x), ..., \phi_n(x)\}$ som danner basisen for $\mathbf{V_h}$. Approksimerer v^{n+1} som en lineær kombinasjon av basisfunksjonene.

$$v^{n+1} \approx v_h = \sum_{j=1}^{n} v_j \phi_j \ .$$

Ved bruk av Galerkins metode benyttes de samme basisfunksjonene for å approksimere vektfunksjonene, $w = \phi_i$ for $i = 1, ..., n$. Det diskre problemet blir

Finn $v_h \in \mathbf{V_h}$ slik at for alle $\phi_i \in \mathbf{V_h}$

$$\alpha(v_h, \phi_i) + \Delta t \theta a(v_h, \phi_i) = \alpha(v^n, \phi_i) - \Delta t(1-\theta) a(v^n, \phi_i) \tag{6.40}$$
$$V(x,0) = I \ .$$

Element formulering

Setter inn for v_h

$$\alpha \sum_{j=1}^{n} v_j \int_\Omega \phi_j \phi_i \ d\Omega + \theta \Delta t \sum_{j=1}^{n} v_j \int_\Omega M \nabla \phi_j \nabla \phi_i \ d\Omega$$
$$= \alpha \int_\Omega v^n \phi_i \ d\Omega - \Delta t(1-\theta) \int_\Omega M \nabla v^n \nabla \phi_i \ d\Omega \quad i = 1, .., n \tag{6.41}$$

hvilket er et system av lineære ligninger som kan skrives på formen

$$Av = b \ , \tag{6.42}$$

hvor

$$A_{ij} = \alpha \int_\Omega \phi_j \phi_i \ d\Omega + \theta \Delta t \int_\Omega M \nabla \phi_j \nabla \phi_i \ d\Omega$$
$$b_i = \alpha \int_\Omega v^n \phi_i \ d\Omega - \Delta t (1-\theta) \int_\Omega M \nabla v^n \nabla \phi_i \ d\Omega \ .$$

For et implisitt Euler skjema, $\theta = 1$, får vi

$$A_{ij} = \alpha \int_\Omega \phi_j \phi_i \ d\Omega + \Delta t \int_\Omega M \nabla \phi_j \nabla \phi_i \ d\Omega$$
$$b_i = \alpha \int_\Omega v^n \phi_i \ d\Omega \ .$$

For et Crank-Nicholson skjema, $\theta = 1/2$, får vi

$$A_{ij} = \alpha \int_\Omega \phi_j \phi_i \ d\Omega + \frac{\Delta t}{2} \int_\Omega M \nabla \phi_j \nabla \phi_i \ d\Omega$$
$$b_i = \alpha \int_\Omega v^n \phi_i \ d\Omega - \frac{\Delta t}{2} \int_\Omega M \nabla v^n \nabla \phi_i \ d\Omega \ .$$

Implementasjon

Program 6 og Program 7 viser et utsnitt av koden vår, hvor resten kan sees i Tillegg A.2. Vi ser hvordan matrisen A og vektoren b skal vektes for de tre ulike metodene. Ved et implisitt Euler skjema ($\theta = 1$) og et Crank-Nicholson skjema($\theta = 1/2$) får vi det lineære systemet $Av = Hv_n$ som skal løses, hvor Hv_n tilsvarer vektoren b. For et TR-BDF2 skjema i tid kan vi ikke bruke θ-regelen, men fremgangsmåten for FEM er ellers lik. Her får vi to lineære systemer som skal løses, dette går frem av Program 7. Steg 1 gir systemet $A_1 v_{mid} = b$, hvor $b = b_1 v_n$. Steg 2 gir systemet $A_2 v = b$, hvor $b = b_2 v_{mid} + b_3 v_n$. Vi setter opp Konjugerte gradienters(CG) metode, se Seksjon 6.5, i begge programmene. Dette skal brukes ved løsning av monodomeneproblemet for hvert tidssteg. $P = Prec(A)$ gir en prekondisjonering av systemet i CG-metoden. Hvis $P = none$ bruker vi CG-metoden uten prekondisjonering.

6.4 ODE-løsere

Operator-splittingen som foretas ved løsning av modellene fører til en reduksjon fra et komplisert system til et lineært PDE-system og et ikke-lineært ODE-system.

Program 6 Løsning av Monodomeneproblemet diskretisert med implisitt Euler og CN

```
mf = MatrixFactory(mesh)
M = mf.computeMassMatrix()
Ai = mf.computeStiffnessMatrix(tf)

if method == 'cn':
   #method = Crank-Nicholson
   a = 1.0; b = 1.0/2.0; c = 1.0; d = 1.0/2.0
else:
   #method = implisitt Euler
   a = 1.0; b = 1.0; c = 1.0; d = 0
A = a*M + b*dt*Ai
H = c*M - d*dt*Ai

P = Prec(A)
cg = ConjugateGradients(A, None, P, TOL, True,
     maxiter=4000, info=True)
```

Program 7 Løsning av Monodomeneproblemet diskretisert med TR-BDF2

```
if method == 'trbdf2':
    #method = TR-BDF2
    gamma = 1 - math.sqrt(2)/2.0
    gamma2 = (1-(2*gamma))/(2*(1-gamma))

    A1 = M + gamma*dt*Ai
    b1 = M - gamma*dt*Ai

    A2 = M + gamma2*dt*Ai
    b2 = (1-gamma2)/(2*gamma)*M
    b3 = (1-(1-gamma2)/(2*gamma))*M

P1 = Prec(A1)
cg1 = ConjugateGradients(A1, None, P1, TOL, True,
      maxiter=4000, info=True)
P2 = Prec(A2)
cg2 = ConjugateGradients(A2, None, P2, TOL, True,
      maxiter=4000, info=True)
```

Uavhengig om vi løser Monodomene- eller Bidomenemodellen, og uavhengig av hvilket tidsdikretiseringsskjema som blir brukt, så vil det å løse det ikke-lineære ODE-systemet være likt.

6.4.1 GRL2

Her ønsker vi å bruke en 2.ordens ODE-løser, GRL2, som er en utvidelse av Rush-Larsen metoden [22]. Skjemaet gitt ved GRL2 er eksplisitt slik at vi kan unngå å måtte løse ikke-lineære algrbraiske ligninger [23].

De fleste cellemembran modeller kan formuleres som et startverdi problem på formen

$$\frac{dy}{dt} = f(y) \qquad\qquad y(t_0) = y_0 \tag{6.43}$$

der y er en vektor. En lokal diagonal lineærisering av $f(y)$ i (6.43) er en viktig del av utvidelsen av Rush-Larsen metoden. Man foretar en dekobling og en lineærisering av problem (6.43) rundt et punkt η

$$u_i' = f_i(\eta) + (u_i - \eta_i)\frac{\partial}{\partial y_i} f_i(\eta) \qquad\qquad u_i(t_n) = \eta_i \tag{6.44}$$

for $i = 1, .., k$. Eksakt løsning av system 6.44 er gitt ved

$$u_i(t) = \eta_i + \frac{a}{b}(e^{b(t-t_n)} - 1) \qquad\qquad i = 1, .., k \tag{6.45}$$

hvor $a = f_i(\eta)$ og $b = \partial f_i(\eta)/\partial y_i$
En approksimasjon av u^{n+1} ved $t = t_n + \Delta t$ kan nåes i to steg. Vi antar en approksimasjon u^n av problem 6.43 er kjent ved tiden $t_n = t_0 + n\Delta t$

1. Løs de k ODE'ene (6.44) med $\eta = u^n$ eksakt opp til $t = t_{n+1/2}$. Løsningene $u^{n+1/2}$ gis ved (6.45).

2. Lar $\bar{u}^{(i)}$ gi den midlertidige løsningsvektoren $u^{n+1/2}$ med den i^{te} komponenten byttet ut med u_i^n

$$\bar{u}^{(i)} = (u_1^{n+1/2}, .., u_{i-1}^{n+1/2}, u_i^n, u_{i+1}^{n+1/2}, .., u_k^{n+1/2}) \tag{6.46}$$

   Løser (6.44) med $\eta = \bar{u}^{(i)}$ opp til $t = t_{n+1}$. Løsningen u^{n+1} er gitt ved (6.45).

6.5 Iterative løsere for lineære systemer

I denne oppgaven vil vi bruke Konjugerte gradienters metode for å løse systemet

$$Av = b \tag{6.47}$$

6.5.1 Konjugerte gradienters metode

I 1952 kom Hestenes & Stifel [24] med en algoritme for å løse lineære systemer på formen $Av = b$. Denne algoritmen ga en metode for å gi en eksakt løsning på systemet på høyst N iterasjoner. N er antall ukjente i systemet, og A må være symmetrisk og positiv definite. Positive definite vil si at indreproduktet $(Lu, u) > 0$ for alle relevante funksjoner u, mens symmetrisk vil si at $(Lu, v) = (u, Lv)$, hvor L er en operator. Algoritmen er som følger:

Konjugerte gradienters algoritme

La $A \in \mathbb{R}^{N,N}$, $b \in \mathbb{R}^N$, $v^0 \in \mathbb{R}^N$, $0 < \epsilon < 1$.

$v = v^0$

$r = b - Av$

$p = r$

$\rho_0 = (r, r)$

$k = 0$

while $\rho_k/\rho_0 > \epsilon$ do

$$\begin{aligned} z &= Ap \\ \gamma &= (p, z) \\ \alpha &= \rho_k/\gamma \\ v &= v + \alpha p \\ r &= r - \alpha z \\ \rho_{k+1} &= (r, r) \\ \beta &= \rho_{k+1}/\rho_k \\ p &= r + \beta p \\ k &= k + 1 \end{aligned}$$

end

Prekondisjonering

Algoritmen er som følger:

Konjugerte gradienters algoritme

La $A \in \mathbb{R}^{N,N}$, $b \in \mathbb{R}^N$, $v^0 \in \mathbb{R}^N$, $0 < \epsilon < 1$. $r = b - Av$

$y = M^{-1}r$

$p = y$

$\rho_0 = (r, y)$

$k = 0$

while $\rho_k/\rho_0 > \epsilon$ do

$$
\begin{aligned}
z &= Ap \\
\gamma &= (p, z) \\
\alpha &= \rho_k/\gamma \\
v &= v + \alpha p \\
r &= r - \alpha z \\
y &= M^{-1}r \\
\rho_{k+1} &= (r, y) \\
\beta &= \rho_{k+1}/\rho_k \\
p &= y + \beta p \\
k &= k + 1
\end{aligned}
$$

end

Hvor M er prekondisjoneringen og $(u, v) = \frac{1}{N}\sum_{i=1}^{N} u_i v_i$ er et innerprodukt. Matrisen M må være symmetrisk og positive definite. Ideen med prekondisjonering er å multiplisere systemet (6.47) med en matrise eller lineær operator M for å oppnå et tilsvarende system

$$MAv = Mb\,. \tag{6.48}$$

Det er viktig at M er en god approksimasjon av A^{-1} og at den er rimelig med hensyn på lagring og evaluering.

Kapittel 7

Resultater

I dette kapittelet skal vi se på resultatene av simuleringene våre. Seksjon 7.1 tar for seg simuleringene av Monodomeneproblemet, seksjon 7.2 tar for seg simuleringene av Bidomeneproblemet og seksjon 7.3 tar for seg simuleringer av Monodomeneproblemet med TNNP som cellemodell.

7.1 Monodomeneproblemet

7.1.1 Konvergens

Når vi skal løse monodomeneproblemet ønsker vi å bruke en 2.ordens metode. En 2.ordens metode er mer nøyaktig enn en 1.ordens metode, og feilen i forhold til vår referanseløsning vil være mindre. Under diskretisering av Monodomenemodellen ble det brukt Strang-splitting for å forenkle problemet. Vi fikk et enklere system bestående av en ODE-del og en PDE-del. For å kunne utnytte at Strang-splitting er en 2.ordens splittemetode, må vi også bruke 2.ordens løsere på våre delproblemer. For ODE-delen ble det bestemt å bruke GRL2, se seksjon 6.4 på side 51. PDE'en ble først diskretisert i tid ved hjelp av θ-regelen. Ved valg av $\theta = 1/2$ oppnår vi et Crank-Nicholson(CN) skjema i tid, hvilket gir et 2.ordens skjema. Velger vi $\theta = 1$ får 1.ordens implisitt Euler. I tillegg ble det valgt å diskretisere i tid ved hjelp av et TR-BDF2 skjema, som også gir en 2.ordens metode. I Tillegg A finner vi noe av koden som er brukt. Tillegg A.1 gir implementasjonskoden for **monodomainsolver.py**. Dette viser hvordan vi kan løse Monodomeneproblemet i 2D. For å løse problemet trenger vi å sette opp systemet $Ax = b$ og konjugerte gradienters metode, og dette gjøres i Tillegg A.2 **BidomainAssemblerAndSolver.py**. BidomainAssemblerAndSolver.py gir deg mulighet til å løse både Monodomeneproblemet og Bidomeneproblemet, avhengig av hvilke data som er gitt.

Før vi tester om løsningen av Monodomeneproblemet gir god nok nøyaktighet, ønsker vi å teste ODE-delen og PDE-delen hver for seg. På denne måten får vi testet om implementeringen er korrekt og at det fungerer på et enkelt problem.

Ved testing av GRL2 ble Aliev-Panfilov [20] valgt som cellemodell. Vi ser på aksjonspotensialet, se figur 3.1, for en enkelt celle over tid, og beregner feilen, se boks (7.1), for løsningene ved T = 200, hvor referanseløsning er beregnet med dt = 0.00195312. Konvergensen er definert som $e_o/e_1 = e_{dt}/e_{dt/2}$. Ut fra tabell 7.1 kan vi se at konvergensen for løsningene er tilnærmet 2.orden, slik at GRL2 oppfører seg som en 2.ordens løser.

Relativ feil

Gitt en verdi v og en approksimert verdi v_{app}:

$$e = \frac{|v - v_{app}|}{|v|}$$

Gitt en vektor $\mathbf{v}$ og en approksimert vektor $\mathbf{v}_{app}$:

$$\mathbf{e} = \frac{||\mathbf{v} - \mathbf{v}_{app}||}{||\mathbf{v}||} \tag{7.1}$$

Varmeledningsligningen, se seksjon 5.2.1, ble testet slik at vi fikk kjørt PDE'problemet alene. Vi velger at initialverdien til transmembranpotensialet skal være uavhengig av y-verdi i gridet. Dermed kan vi bruke en propagerende vegg som beveger seg gjennom gridet fra $x = 0$ til $x = 1$. Vi velger følgende initialverdi:

$$v(x, 0) = -85(1 - e^{-200x^2}) \ .$$

Denne initialverdien vil brukes gjennom resten av oppgaven. Toleranse for feilen til $Ax = b$ er satt til TOL = 1.e-8. Referanseløsningen her er beregnet med dt = 0.015625, og feilen er beregnet ved T = 10ms. Vi testet først varmeledningsligningen med bruk av 1.ordens implisitt Euler, og tabell 7.2 viser at konvergensen her er tilnærmet 1.orden. Siden en 1.ordens metode ikke er tilstrekkelig med tanke på nøyaktighet, går vi videre med å teste varmeledningsligningen løst med et Crank-Nicholson skjema i tid og tilslutt løst med et TR-BDF2 skjema i tid. Tabell 7.2 viser at konvergensen for både Crank-Nicholson og TR-BDF2 er tilnærmet lik 4, det vil si 2.orden. Dette viser at våre implementerte skjema oppfører seg som de skal.

Siden tester for ODE-løseren og for varmeledningsligningen viser 2.orden, kan vi gå videre til å teste om Monodomeneproblemet gir 2.ordens løsninger. Vi starter

dt	e	e_0/e_1
0.125	0.012453	
0.0625	0.003421	3.6399
0.03125	0.000892	3.8344
0.015625	0.000225	3.9599
0.0078125	5.422e-05	4.1555
0.00390625	1.090e-05	4.9751

Tabell 7.1: Feil og konvergens for aksjonspotensialet i en enkelt celle beregnet ved GRL2

	Euler		CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.001327		0.000113		5.521e-05	
0.5	0.001552	2.0145	2.802e-05	4.0373	1.362e-05	4.0541
0.25	0.000754	2.0579	6.909e-06	4.0563	3.380e-06	4.0292
0.125	0.000353	2.1384	1.710e-06	4.0400	8.345e-07	4.0502
0.0625	0.000151	2.3309	4.091e-07	4.1803	2.071e-07	4.0294
0.03125	5.047e-05	2.9985	8.700e-08	4.7019	6.329e-08	3.2725

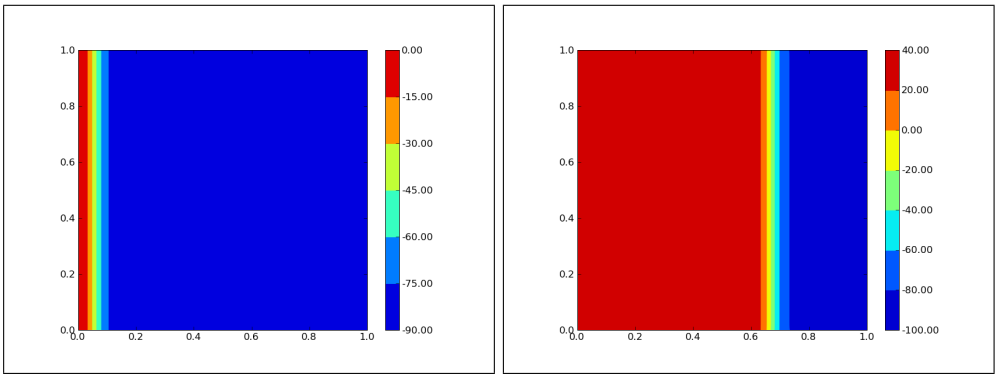
Tabell 7.2: Feil og konvergens for varmeledningsligningen

med å splitte Monodomeneproblemet både med Godunov og Strang splitting, deretter diskretiserer vi i tid med implisitt Euler, Crank-Nicholson og med TR-BDF2. For ODE-delen fortsetter vi å bruke Aliev-Panfilovs cellemodell og GRL2 som ODE-løser. Vi beregner feilen ved T = 10ms, og referanseløsningen er beregnet med dt = 0.015625. Monodomeneproblemet løses på enhetskvadrat med gridoppløsning 500 i x-retning og 10 i y-retning, se program 8. Vi vil se på propagering i x-retning, og

Program 8 Gridet for løsning av Monodomeneproblemet

```
from dolfin import *
K=500;L=10
# Make the mesh
mesh =  UnitSquare(K,L)
```

velger av den gunn en høyere tetthet her. Samtidig kan vi spare tid på beregningene ved å velge lav tetthet av noder i y-retning. Resultatet av en typisk kjøring vises i figur 7.1. Her ser vi transmembranpotensialet ved T = 0ms og T = 10ms. Tabell 7.3 viser at Monodomeneproblemet løst med Strang-splitting, samt CN eller TR-BDF2, gir en konvergens tilnærmet 4, det vil si 2.orden. Ut i fra tabell 7.3 og tabell 7.4 ser vi at henholdsvis Strang-splitting og Godunov-splitting kombinert med implisitt Euler gir en konvergens tilnærmet 1.orden, hvilket var forventet. Samtidig viser ta-



(a) T = 0ms (b) T = 10ms

Figur 7.1: Transmembranpotensialet i gridet

	Euler		CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.587834		0.432951		0.322506	
0.5	0.411201	1.4296	0.047918	9.0353	0.025661	12.5680
0.25	0.244785	1.6798	0.043848	1.0928	0.015782	1.6260
0.125	0.128992	1.8977	0.010541	4.1598	0.004457	3.5409
0.0625	0.058889	2.1904	0.002535	4.1574	0.001099	4.0566
0.03125	0.020261	2.9066	0.000511	4.9615	0.000222	4.9461

Tabell 7.3: Feil og konvergens for Monodomeneproblemet løst med Strang-splitting

bell 7.4 at Godunov-splitting kombinert med CN gir en konvergens på oppunder 2.orden, som er noe bedre enn forventet, mens Godunov-splitting kombinert med TR-BDF2 gir konvergens på rundt 3.

Ut fra disse testene har vi sett at GRL2 oppfører seg som en 2.ordens løser. Varmeledningsligningen løst med CN og TR-BDF2 som tidsdiskretisering gir 2.ordens løsning, mens implisitt Euler gir 1.ordens løsning. Videre ble det vist at feilen for løsningene av Monodomeneproblemet ble minst ved bruk av Strang-splitting og TR-BDF2 som tidsdiskretisering.

7.1.2 Toleranse for feil i den lineære likningsløseren

Vi skal her se på betydningen av toleranse(TOL)-verdi for feil i den lineære løserlikningen ($Ax = b$), og om dette har noe å si på konvergensen ved løsning av varmeledningsligningen. Det er ønskelig å bruke minst mulig tid ved kjøring, og en for liten TOL-verdi kan gi unødvendig CPU-tid. Varmeledningsligningen ble løst med

	Euler		CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.285763		nan		0.396628	
0.5	0.240016	1.1906	0.238856	nan	0.240148	1.6516
0.25	0.192788	1.2450	0.110013	2.1712	0.080491	2.9835
0.125	0.115244	1.6729	0.026259	4.1896	0.020512	3.9241
0.0625	0.056341	2.0455	0.006578	3.9917	0.005609	3.6567
0.03125	0.020111	2.8014	0.001717	3.8319	0.001641	3.4186

Tabell 7.4: Feil og konvergens for Monodomeneproblemet løst med Godunov-splitting

	TOL=1.e-5		TOL=1.e-8		TOL=1.e-16	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	5.858e-05		5.521e-05		5.521e-05	
0.5	2.128e-05	2.7526	1.362e-05	4.0541	1.362e-05	4.0539
0.25	7.965e-05	0.2672	3.380e-06	4.0292	3.374e-06	4.0368
0.125	1.627e-05	4.8957	8.345e-07	4.0502	8.310e-07	4.0601
0.0625	1.160e-05	1.4023	2.071e-07	4.0294	1.976e-07	4.2063
0.03125	6.853e-06	1.6931	6.329e-08	3.2725	3.948e-08	5.0036

Tabell 7.5: Feil og konvergens for varmeledningsligningen løst med TR-BDF2

både implisitt Euler, CN og TR-BDF2 skjema, og feilen ble beregnet ved T = 10ms. Referanseløsningen er beregnet med dt = 0.015625. Ved bruk av TR-BDF2 skjema ved løsning av varmeledningsligningen er vi, ut ifra tabell 7.5, avhengig av liten nok TOL-verdi for at vi skal få tilfredsstillende konvergens ved kjøring lenger enn 10ms. CN, se tabell 7.6, klarer seg med en større TOL-verdi enn det TR-BDF2 gjør. Implisitt Euler derimot er ikke like avhengig av en lav TOL-verdi, se tabell 7.7. Dette henger sammen med at feilen for løsningen i forhold til referanseløsningen er stor. Figur 7.2 viser et log-log plot av feilen mot dt. Her er tydelig å se at feilen gitt ved implisitt Euler er tilnærmet lik uavhengig av valg av TOL-verdi. TOL-verdi = 1e-05 vil for både CN og TR-BDF2 gi feil som ikke følger 2.ordens konvergens. TOL-verdi=1e-08 vil gi gode resultater for CN , men vi får et lite utslag for TR-BDF2 når dt blir liten. TOL-verdi = 1e-16 gir gode resultater for alle metodene.
Vi tester i tillegg betydningen av TOL-verdi ved løsning av Monodomeneproblemet. Vi beregner feilen ved T = 10ms, hvor referanseløsningen vi bruker er beregnet med dt = 0.015625 med TOL = 1e-16. I tabell 7.8 ser vi at feilen for løsningen av Mondomeneproblemet er tilnærmet lik uavhengig om vi bruker TOL = 1e-08 eller TOL = 1e-16. Dette betyr at vi kan sette TOL=1e-08 for fremtidige tester.

	TOL=1.e-5		TOL=1.e-8		TOL=1.e-16	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.000114		0.000113		0.000113	
0.5	3.055e-05	3.7383	2.802e-05	4.0373	2.802e-05	4.0374
0.25	3.602e-05	0.8482	6.909e-06	4.0562	6.909e-06	4.0563
0.125	5.869e-05	0.6138	1.710e-06	4.0400	1.707e-06	4.0474
0.0625	9.068e-06	6.4719	4.091e-07	4.1803	4.064e-07	4.2000
0.03125	5.473e-06	1.6568	8.701e-08	4.7019	8.128e-08	5.0000

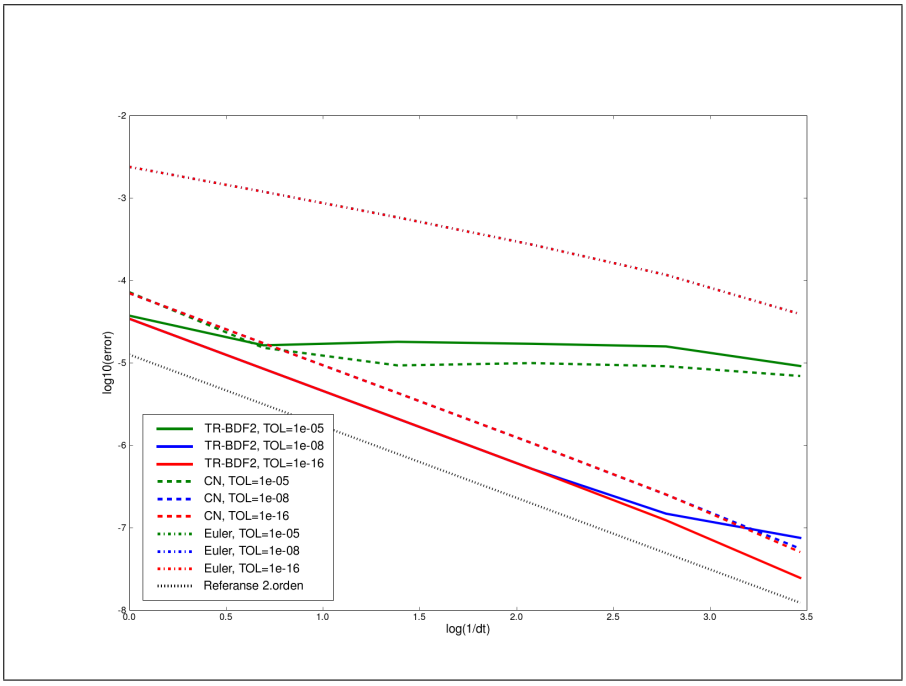
Tabell 7.6: Feil og konvergens for varmeledningsligningen løst med CN

	TOL=1.e-5		TOL=1.e-8		TOL=1.e-16	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.003127		0.003127		0.003127	
0.5	0.001553	2.0139	0.001552	2.0145	0.001552	2.0145
0.25	0.000754	2.0601	0.000754	2.0579	0.000754	2.0579
0.125	0.000364	2.0696	0.000353	2.1384	0.000353	2.1384
0.625	0.000165	2.2041	0.000151	2.3309	0.000151	2.3309
0.0315	5.058e-05	3.2665	5.047e-05	2.9985	5.047e-05	2.9985

Tabell 7.7: Feil og konvergens for varmeledningsligningen løst med implisitt Euler

	TOL=1.e-8	TOL=1.e-16
dt	e	e
1	0.464777	0.464777
0.5	0.296472	0.296472
0.25	0.102740	0.102740
0.125	0.027593	0.027593
0.625	0.00690303	0.00690306
0.0315	0.00141912	0.00141915

Tabell 7.8: Feil for Monodomeneproblemet løst med TR-BDF2



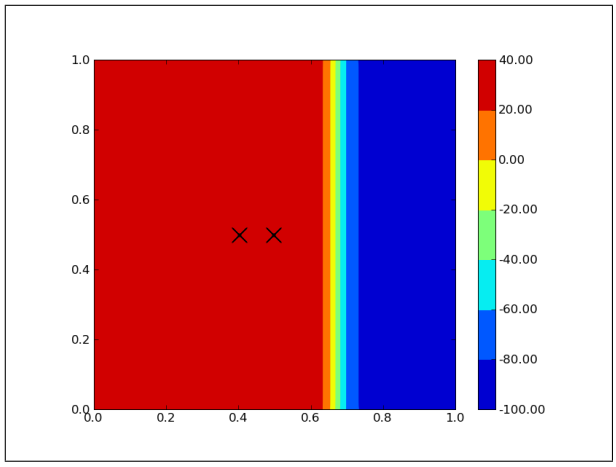
Figur 7.2: Virkning av ulike TOL-verdier ved T = 10ms

7.1.3 Propageringshastighet

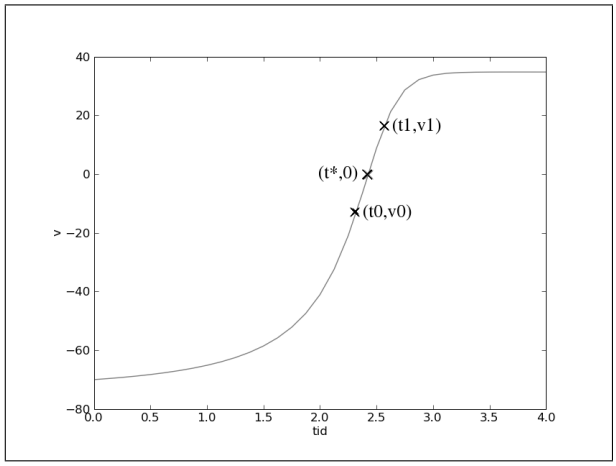
Verdier for transmembranpotensialet ble vurdert i noen punkter i gridet for å finne hvor fort aksjonspotensialet propagerte, se figur 7.3(a). Vi velger å regne ut ved hvilke tidspunkter transmembranpotensialet er lik 0 i våre to punkter. Dette kan gjøres ved hjelp av topunktsformelen, hvilket er:

$$v(t) = \frac{v_1 - v_0}{t_1 - t_0}(t - t_0) + v_0 \ ,$$

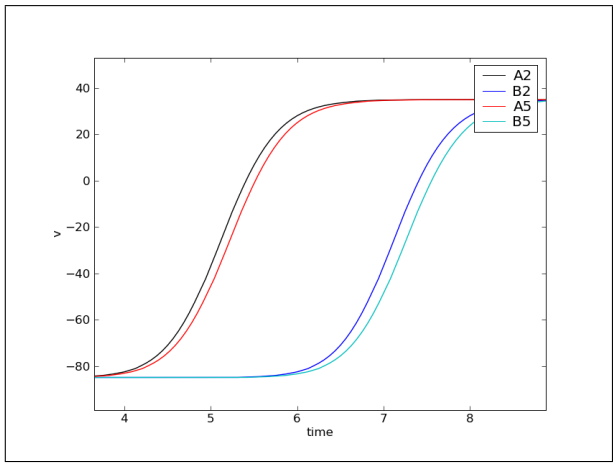
hvor v_0, v_1, t_0 og t_1 er kjente verdier. Figur 7.3(b) gir en definisjon av punktene. Propageringshastigheten får vi ved ta avstanden mellom punktene i gridet og dividere på tidsdifferansen ($t^*_{(0.5,0.5)} - t^*_{(0.4,0.5)}$; hvor $_{(x,y)}$ angir punkt i gridet). Figur 7.3(c) viser aksjonspotensialet i to punkter, A og B, hvor 2 og 5 angir henholdsvis $dt = 2^{-2}$ og $dt = 2^{-5}$. Propageringshastigheten beregnet ved dt = 2^{-2} gis ved $\Delta x/(t^*_{B2} - t^*_{A2})$, hvor Δx er avstanden mellom A og B i gridet. Tabell 7.9 og tabell 7.10 gir propageringshastighet og konvergensraten for hastigheten, hvor referanseløsning er beregnet med dt = 0.015625 og henholdsvis CN og TR-BDF2 er brukt som tidsdiskretiseringsskjema. Vi ser at propageringshastigheten også konvergerer mot 2.orden hvor vi får noe mer nøyaktige løsninger ved bruk av TR-BDF2.



(a) Punkter i gridet for beregning av propageringshastighet



(b) Definisjon av punkter brukt i topunktsformelen



(c) Aksjonspotensialet for punkt A og B ved dt = 2^{-2} og dt = 2^{-5}

Figur 7.3: Beregning av propageringshastighet

dt	propageringshastighet	e	e_1/e_2
0.5	0.025240	0.022262	
0.25	0.026072	0.009936	2.2405
0.125	0.025882	0.002596	3.8276
0.0625	0.025833	0.000674	3.8544
0.03125	0.025818	0.000123	5.4807

Tabell 7.9: Propageringshastighet og konvergensrate, CN

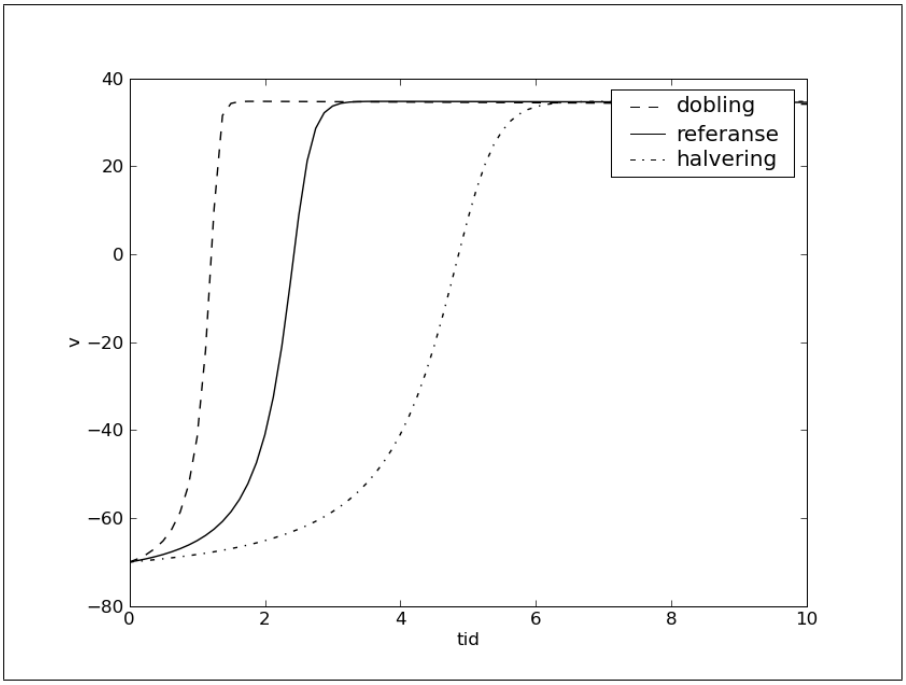
dt	propageringshastighet	e	e_1/e_2
0.5	0.025910	0.003705	
0.25	0.025927	0.004384	0.8451
0.125	0.025837	0.000877	4.9959
0.0625	0.025819	0.000198	4.4374
0.03125	0.025815	4.027e-05	4.4664

Tabell 7.10: Propageringshastighet og konvergensrate, TR-BDF2

7.1.4 Upstroke velocity

Upstroke velocity er raten til endring av transmembranpotensiale i depolariseringsfasen. Ved å endre på parametere i cellemodellen kan upstroke velocity endres. En endring i upstroke velocity fører til en endring av propageringshastighet. Figur 7.4 viser plot av aksjonspotensialet i en enkelt celle med ulike upstroke velocity verdier.

Vi tester først om ulike upstroke velocity rater i en enkelt celle har en påvirkning på konvergensen. Referanseløsning er beregnet med dt = 0.015625. Det viser seg at lav upstroke rate gir bedre konvergens av løsningene og at disse er mer nøyaktige enn ved en høyere upstroke rate, se tabell 7.11. En dobling av upstroke raten i forhold til vår referanse gir noe dårligere konvergens og mer unøyaktige løsninger.
Hvis vi tester upstroke velocity på Monodomeneproblemet, hvor TR-BDF2 er brukt som tidsdiskretiseringsskjema, vil vi også her få at en endring i upstroke raten påvirker nøyaktigheten til løsningene og konvergensen av dem, se tabell 7.12. En dobling av upstroke raten fører til ustabile løsninger og vi må ha liten dt før løsningene blir stabile. En halvering av upstroke raten fører også her til mer nøyaktige løsninger. Upstroke rate påvirker i tillegg propageringshastigheten. Ved en halvering av upstroke velocity får vi en minkning i propageringshastighet, mens ved en dobling av upstroke velocity får vi en økning i propageringshastighet, se tabell 7.13. Tabell 7.14 viser konvergensen for feilen til propageringshastighetene ved ulike upstroke rater. Feilen for propageringshastigheten i forhold til referansen vil bli betydelig mindre ved å halvere upstroke raten. Vi kan se at propageringshastigheten konver-



Figur 7.4: Virkning på aksjonspotensialet i en enkelt celle ved ulike upstroke velocities

gerer mot referansehastigheten med 2.orden.

Vi har sett at halvering av upstroke velocity fører til at vi får minkning i propageringshastighet og mer nøyaktige løsninger i forhold til referanseløsningen. En dobling av upstroke velocity fører til aksjonspotensialet propagerer raskere gjennom domenet, og at vi samtidig får mer ustabile løsninger slik at vi er avhengig av mindre dt. I tillegg blir løsningene mindre nøyaktige i forhold til referanseløsningen.

	referanse		halvering		dobling	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.008047		0.000463		0.369277	
0.5	0.000550	14.6239	0.000128	3.6216	0.011343	32.5543
0.25	0.000152	3.6172	3.755e-05	3.4061	0.000793	14.3121
0.125	4.431e-05	3.4329	9.961e-06	3.7693	0.000219	3.6243
0.625	1.133e-05	3.9099	2.451e-06	4.0635	6.150e-05	3.5561
0.0315	2.341e-06	4.8407	4.9789e-07	4.9236	1.321e-05	4.6550

Tabell 7.11: Feil og konvergens for transmembranpotensialet i en enkelt celle løst med GRL2, T = 25ms, ved ulike upstroke rater

	referanse		halvering		dobling	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.496029		0.018082		1.629399	
0.5	0.035297	14.0532	0.011771	1.5361	1.009069	1.6148
0.25	0.020447	1.7262	0.003396	3.4663	0.007255	139.0790
0.125	0.005959	3.4312	0.000867	3.9179	0.008390	0.8648
0.625	0.001463	4.0725	0.000208	4.1683	0.003327	2.5218
0.0315	0.000294	4.9792	4.165e-05	4.9927	0.000771	4.3159

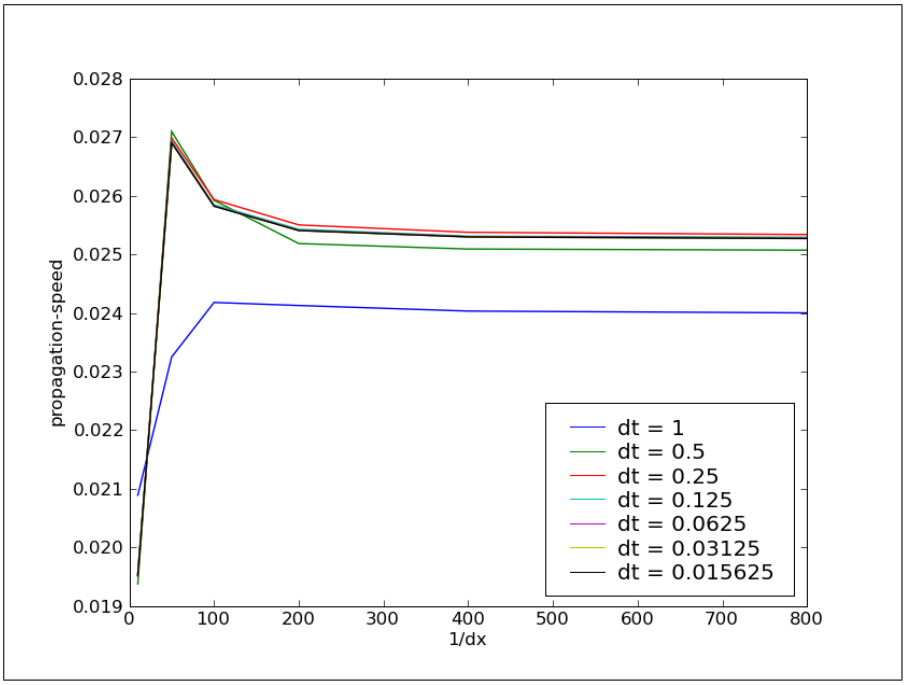
Tabell 7.12: Feil og konvergens for Monodomeneproblemet, T=25ms, ved ulike upstroke rater

dt	referanse	halvering	dobling
0.5	0.025910	0.018109	0.033457
0.25	0.025927	0.018070	0.036965
0.125	0.025837	0.018066	0.037315
0.0625	0.025819	0.018063	0.037201
0.03125	0.025815	0.018063	0.037177
0.015625	0.025814	0.018063	0.037171

Tabell 7.13: Propageringshastighet gitt ved ulike upstroke rater for Monodomenproblemet

	referanse		halvering		dobling	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
0.5	0.003705		0.002574		0.099928	
0.25	0.004384	0.8451	0.000440	5.8499	0.005546	18.0182
0.125	0.000877	4.9959	0.000190	2.3100	0.003853	1.4393
0.625	0.000198	4.4374	5.033e-05	3.7842	0.000783	4.9227
0.0315	4.427e-05	4.4664	1.200e-05	4.1927	0.000152	5.1492

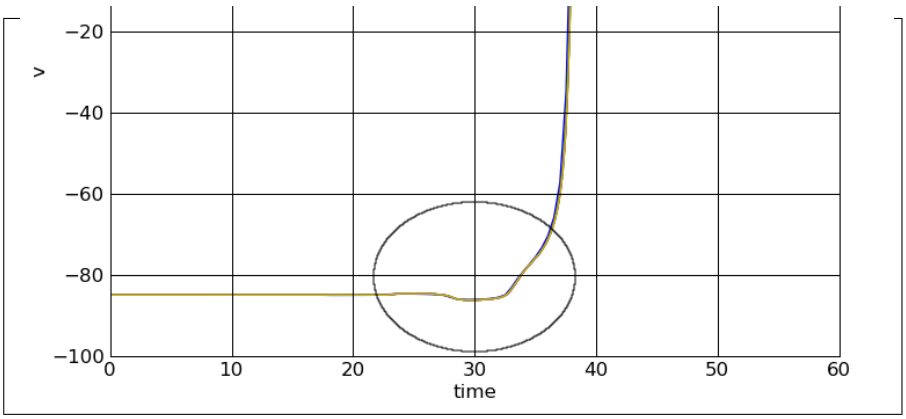
Tabell 7.14: Feil og konvergens for propageringshastigheten ved ulike upstroke rater



Figur 7.5: Propageringshastighet som funksjon av Δx for ulike Δt

7.1.5 Gridoppløsning

Vi skal her undersøke betydning av gridoppløsning, og om dette hadde en påvirkning på propageringshastigheten. Størrelsen på Δx ble variert, mens Δy ble holdt konstant på $1/10$. Ved $\Delta x > 1/200$cm fikk vi større variasjoner i propageringshastigheten enn for $\Delta x < 1/200$cm. Det vil si at propageringshastigheten er tilnærmet lik uavhengig av valg av Δx, såfremt $\Delta x < 1/200$cm = 0.05mm, se figur 7.5. I tillegg kunne det observeres at ved lav gridoppløsning fikk vi verdier for transmembranpotensialet som lå under v_{rest}, se figur 7.6.



Figur 7.6: Verdier for transmembranpotesialet under v_{rest}, gridoppløsning (10,10)

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	6195	6.84	3754	6.20	6857	10.71
0.5	8553	9.69	6359	10.09	9067	11.07
0.25	11583	13.13	8265	11.59	11582	13.06
0.125	15354	17.60	10250	16.35	14721	16.86
0.625	19976	22.67	13508	17.46	18890	21.87
0.0315	25722	29.15	17380	22.46	23755	28.12
0.015625	32428	38.31	21794	28.83	30647	38.79

Tabell 7.15: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ uten prekondisjonering

7.1.6 CPU-tid

Tiden det tar å løse Monodomeneproblemet er av betydning ved valg av tidsdiskretiseringsskjema. Selv om TR-BDF2 viser seg å være mer nøyaktig enn CN ved en gitt oppløsning, brukes det også lenger tid på å løse Monodomeneproblemet når vi benytter oss av TR-BDF2. Tiden som brukes av GRL2 for å løse ODE-delen fra Strang-splittingen vil være lik uavhengig av hvilket tidsdiskretiseringsskjema som brukes. Dermed skal vi bare se på CPU-tid for PDE-delen. PDE-delen i Monodomeneproblemet løses ved hjelp av konjugerte gradienters metode , se seksjon 6.5 på side 54, hvilket er en iterativ metode. Det lineære systemet $Ax = b$ løses iterativt til normen til residualen er mindre enn en TOL-verdi. Denne TOL-verdien ble tidligere satt til å være 1.e-8.

Tabell 7.15 viser CPU-tidsbruken ved løsning av PDE-delen i Monodomeneproblemet over et intervall på T = 10ms. Ved bruk av TR-BDF2 brukes det noe mer CPU-tid og antall iterasjoner N er høyere enn ved CN som tidsdiskretisering. Implisitt Euler er på sin side like tidkrevende som TR-BDF2, og bruker noen flere iterasjoner N. En måte å få konjugerte gradienters metode til å bruke færre iterasjoner, og med andre ord kortere tid, er å benytte seg av prekondisjonering. Dette vises i tabell 7.16 hvor vi kan se at antall iterasjoner har gått betraktelig ned for alle de tre metodene i forhold til å løse $Ax = b$ uten prekondisjonering. Prekondisjonering halverer nesten tidsbruken ved mindre dt, mens antall iterasjoner er omtrent 5-6 ganger mindre. Det er verdt å merke seg at tiden per iterasjon ved prekondisjonering er høyere enn tiden per iterasjon uten prekondisjonering, men systemet $Ax = b$ løses allikvel totalt raskere.

Det vanlige er å bruke v_{n-1} som initialverdi, altså $x_{start} = v_{n-1}$, når systemet $Ax = b$

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	1076	3.33	890	2.71	1223	4.35
0.5	1390	4.27	1074	3.89	1540	4.73
0.25	1939	5.98	1404	4.30	2001	6.29
0.125	2633	8.22	1842	5.74	2643	8.52
0.625	3512	11.26	2406	7.77	3379	11.48
0.0315	4489	14.88	3206	11.64	4521	17.16
0.015625	5828	21.12	4483	16.15	6446	25.37

Tabell 7.16: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ med prekondisjonering

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	6754	8.00	4019	4.56	7412	9.30
0.5	9629	10.80	6833	7.83	10257	12.85
0.25	13444	15.08	9396	10.46	14288	18.07
0.125	18828	21.07	13150	14.77	19985	25.12
0.625	26333	30.11	18458	21.12	28233	35.59
0.0315	36932	42.09	26087	31.29	40240	48.94
0.015625	52179	61.68	37294	44.53	58020	67.70

Tabell 7.17: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ uten prekondisjonering, $x_{start} = 0$

løses. Velger vi i stedet å bruke $x_{start} = 0$ starter vi med et dårligere gjett på initialverdi. Dette burde medføre at konjugerte gradienters metode vil bruke flere iterasjoner. Tabell 7.17 og tabell 7.18 viser antall iterasjoner og CPU-tid ved en slik initialverdi, henholdsvis uten og med prekondisjonering. Både antall iterasjoner og CPU-tid øker noe. Implisitt Euler ligger noe under TR-BDF2 i tidsbruk og antall iterasjoner N, mens begge ligger en del over CN i tidsbruk og iterasjoner. Valg av initialverdi er nok ikke forklaringen på den store forskjellen mellom TR-BDF2 og CN. Siden TR-BDF2 er et 2-(indre)steg tidsdiskretiseringsskjema, se seksjon 6.1.4 på side 35, så må det løses 2 systemer $Ax = b$ for hvert tidssteg. Dette er med på å gi en naturlig forklaring på at antall iterasjoner er høyere ved TR-BDF2 enn ved CN. Tiden per iterasjon for CN og TR-BDF2 er tilnærmet lik.

Legger vi tidsbruken til grunn som en faktor ved valg av tidsdiskretiseringsskjema, kan vi se at vi for CN kan bruke en dt som er halvparten så stor som en gitt dt ved TR-BDF2 uten at vi bruker nevneverdig lenger tid. I seksjon 7.1.1 så vi at feilen for løsningene av Monodomeneproblemet ble minst ved bruk TR-BDF2 som

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	1139	3.47	893	3.43	1276	3.95
0.5	1602	5.56	1156	4.09	1740	5.43
0.25	2255	6.86	1566	5.58	2401	8.35
0.125	3139	9.71	2240	7.94	3448	12.12
0.625	4480	14.28	3200	11.66	5100	17.91
0.0315	6400	20.92	4763	17.29	7625	29.55
0.015625	9529	32.10	7021	26.52	11463	40.96

Tabell 7.18: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ med prekondisjonering, $x_{start} = 0$

tidsdisikretisering. Ser vi derimot i tabell 7.3 og legger til grunn at vi kan velge $dt_{CN} = \frac{1}{2}dt_{TR-BDF2}$ og sammenligner feilen ved disse to dt, så ser vi at feilen vil bli mindre ved å velge CN som tidsdiskretisering.

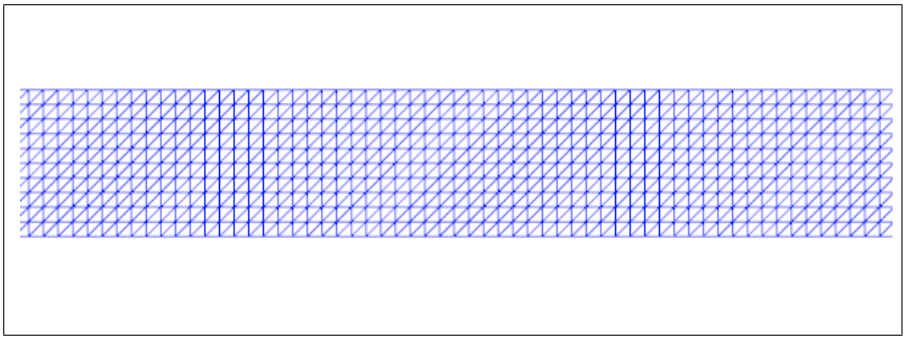
7.2 Bidomeneproblemet

7.2.1 Konvergens

På lik linje som når vi løser Monodomeneproblemet ønsker vi å bruke en 2.ordens metode når vi løser Bidomeneproblemet. Bidomenemodellen ble i seksjon 6.3.1 på side 42 diskretisert med Strang-splitting og med θ-regelen i tid. I tillegg til implisitt Euler og CN velger vi å teste med diskretiseringsskjemaet TR-BDF2. Strang-splittingen gir oss to undersystemer å løse, og ODE-systemet er likt for både Monodomene- og Bidomeneproblemet. I seksjon 7.1.1 på side 57 ble det vist at vi hadde en 2.ordens ODE-løser, GRL2, slik at vi her kan holde oss til å vise 2.orden for PDE-systemet av Strang-splittingen. Det ble også vist at varmeledningsligningen diskretisert med både CN og TR-BDF2 gir 2.orden, samt at implisitt Euler ga 1.orden. Dermed kan vi gå rett til å teste disse tre tidsdiskretiseringsskjemaene på Bidomeneproblemet.

Program 9 Gridet for løsning av Bidomeneproblemet

```
from dolfin import *
m = 10;
n = 50*m ;
mesh = UnitSquare(n,m)
C = mesh.coordinates();
C[:,0] = 50*C[:,0];
C[:,1] = C[:,1];
```



Figur 7.7: Utsnitt av grid (500,10)

	Euler		CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.031956		0.004622		0.004894	
0.5	0.008824	3.6216	0.001012	4.5656	0.000980	4.9920
0.25	0.003303	2.6718	0.000239	4.2374	0.000232	4.2305
0.125	0.001327	2.4891	5.861e-05	4.0758	5.678e-05	4.0819
0.0625	0.000552	2.4038	1.388e-05	4.2221	1.355e-05	4.2228
0.03125	0.000184	3.0056	2.767e-06	5.0181	2.684e-06	5.0097

Tabell 7.19: Feil og konvergens for Bidomeneproblemet

Vi løser Bidomeneproblemet over et tidsrom på T = 5ms og beregner feilen. Referanseløsningen er beregnet med dt = 0.015625. Gridoppløsningen for løsning av Bidomeneproblemet er satt til en avlang boks (500,10), hvor hvert løsningsområde/boks i gridet er kvadratisk, se program 9 og figur 7.7. Tabell 7.19 viser feil i forhold til referanseløsningen og konvergens for henholdsvis Euler, CN og TR-BDF2. For Euler får vi en konvergens på 1.orden. Tidsdiskretiseringsskjemaene CN og TR-BDF2 gir en en konvergens tilnærmet 2.orden, hvilket var forventet, og de er i tillegg tilnærmet like nøyaktige.

7.2.2 CPU-tid

Slik som for Monodomeneproblemet ønsker vi nå kun å se på CPU-tiden for PDE-delen av Strang-splittingen, da tiden som brukes av GRL2 er uavhengig av hvilket tidsdiskretiseringskjema som brukes. PDE'ene løses iterativt ved hjelp av konjugerte gradienters metode. Tabell 7.20 viser CPU-tidsbruken ved løsning av PDE-delen i Bidomeneproblemet over et tidsintervall på T = 5ms. Det er tydelig at konjugerte gradienters metode med TR-BDF2 tidsdiskretiseringen bruker betraktelig flere iterasjoner for å løse systemet enn ved CN tidsdiskretisering. Spesielt for store dt er forskjellen mellom antall iterasjoner for TR-BDF2 og CN høy. Ved mindre dt jev-

nes denne forskjellen noe ut, men med hensyn til antall iterasjoner og CPU-tid vil det være lønnsomt med en CN tidsdiskretisering. Implisitt Euler ligger på omtrent samme nivå som CN med hensyn på både antall iterasjoner og tidsbruk. Vi kan også bruke prekondisjonering slik at konjugerte gradienters metode skal bruke færre iterasjoner, og dermed kortere tid. Dette vises i tabell 7.21. Antall iterasjoner ved prekondisjonering er betraktelig mindre enn uten prekondisjonering. Dermed følger det i tillegg at tidsbruken har minket betraktelig ved prekondisjonering, men på lik linje med Monodomeneproblemet er tiden per iterasjon ved prekondisjonering høyere en tiden per iterasjon uten prekondisjonering. Det at TR-BDF2 er et 2-(indre)steg tidsdiskretiseringsskjema fører til at det løses 2 systemer $Ax = b$ for hvert tidssteg, og for hvert av disse to systemene bruker konjugerte gradienters metode noe færre iterasjoner pr tidssteg enn for CN, men samlet blir antall iterasjoner pr tidssteg høyere for TR-BDF2.

Vi vil også for Bidomeneproblemet teste om valg av initialverdi når systemet $Ax = b$ løses er av betydning. Det vanlige er å starte med $x_{start} = v_{n-1}$, men vi velger her å teste for $x_{start} = 0$. Dette er et dårligere gjett på initialverdi og burde medføre at konjugerte gradienters metode vil bruke flere iterasjoner. Tabell 7.22 og tabell 7.23 viser antall iterasjoner og CPU-tid for løsning av $Ax = b$ henholdsvis uten og med prekondisjonering med $x_{start} = 0$. Uten prekondisjonering er tidsbruken ved implisitt Euler, spesielt for små dt, noe lavere enn for CN. Antall iterasjoner N er derimot omtrent lik for de to metodene. TR-BDF2 ligger noe høyere både når det gjelder antall iterasjoner og tidsbruk. Implisitt Euler gjør det også best, med tanke på både CPU og N, når vi benytter prekondisjonering av systemet. I tillegg er antall iterasjoner for alle metodene noe høyere enn ved $x_{start} = v_{n-1}$, spesielt for små dt, og dermed også tidsbruken. Startverdien gir derimot ikke like stort utslag når prekondisjonering brukes som det det gjør uten. Uten prekondisjonering så øker både antall iterasjoner og tidsbruk meget ved valg av initialverdi lik 0. Initialverdien for u er satt konstant under alle testene, og er ikke blitt endret.

Den høye CPU-tiden ved bruk av TR-BDF2 som tidsdiskretiseringsskjema fører til at vi kan bruke en dt ved CN-diskretiseringen som er halvparten så stor som en gitt dt ved TR-BDF2 uten at tidsbruken økes. Implisitt Euler gjør det også meget bra med tanke på antall iterasjoner og tidsbruk, men skal vi også ta hensyn til nøyaktighet, så vil CN være et bedre valg for tidsdiskretisering. Prekondisjonering av systemet $Ax = b$ er også nødvendig for å holde CPU-tiden nede, mens valg av initialverdi er ikke av like stor betydning hvis vi allerede har sørget for å bruke prekondisjonering av systemet.

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	3333	27.64	3092	25.62	9725	80.42
0.5	6290	51.66	6021	54.28	19484	173.35
0.25	12365	106.88	12494	121.97	33727	286.97
0.125	23857	226.00	24168	217.03	47328	420.97
0.625	38646	404.19	39075	336.05	59237	642.14
0.0315	54478	534.48	54989	542.42	69789	659.70
0.015625	70800	689.50	71067	700.05	77551	803.05

Tabell 7.20: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ uten prekondisjonering for Bidomeneproblemet

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	70	0.96	77	1.20	136	1.80
0.5	120	1.60	128	1.92	223	3.25
0.25	207	2.78	212	3.27	384	5.67
0.125	365	5.00	372	5.76	665	9.63
0.625	645	8.94	648	10.12	1126	16.69
0.0315	1130	16.38	1129	17.92	2070	32.48
0.015625	1958	29.06	1958	34.56	3346	55.36

Tabell 7.21: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ med prekondisjonering for Bidomeneproblemet

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	3345	28.76	3348	30.46	10100	103.10
0.5	6664	61.74	6613	61.07	27933	251.99
0.25	15429	146.36	15422	148.99	76730	754.64
0.125	43342	395.32	43374	511.72	197776	2004.47
0.625	122027	1061.18	122265	1218.52	436197	4664.71
0.0315	331895	2677.56	332863	3277.49	763357	6882.33
0.015625	811649	6597.89	814787	8734.04	1239671	9994.22

Tabell 7.22: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ uten prekondisjonering for Bidomeneproblemet, $x_{start} = 0$

	Euler		CN		TR-BDF2	
dt	N	tid	N	tid	N	tid
1	75	1.49	80	1.35	112	3.15
0.5	140	2.32	142	2.14	250	7.17
0.25	260	3.47	263	4.28	304	9.08
0.125	480	6.58	480	7.47	840	17.74
0.625	889	14.50	892	13.99	1601	45.77
0.0315	1759	24.14	1759	27.26	3182	91.56
0.015625	3200	47.35	3200	51.18	5752	177.02

Tabell 7.23: Antall iterasjoner N og CPU-tid i ms for løsning av $Ax = b$ med prekondisjonering for Bidomeneproblemet, $x_{start} = 0$

7.3 Realistisk cellemodell

Vi ønsker å teste monodomeneproblemet med en ny og mer realistisk cellemodell. Vi velger en TNNP cellemodell av ten Tusscher et al., se seksjon 4.5.3 på side 18. Nye konduktivitetsverdier og verdi for χ har blitt brukt for denne delen av oppgaven, se tabell 7.24.

χ	1400 cm^{-1}
σ_l^i	1.7 mS/cm
σ_t^i	0.19 mS/cm
σ_l^e	6.2 mS/cm
σ_t^e	2.4 mS/cm

Tabell 7.24: Verdier for χ og konduktivitet

7.3.1 Konvergens

Først ønsker vi å se på aksjonspotensialet i en enkelt celle over tid. Vi bruker GRL2 som ODE-løser, og beregner feilen ved T = 200ms og referanseløsningen er beregnet med dt = 0.000976562. Ut i fra tabell 7.25 kan vi se at konvergensen for løsningene er tilnærmet 2.orden når dt blir tilstrekkelig liten. Dermed kan vi også forvente at vi trenger en tilstrekkelig liten dt for å oppnå 2.ordens konvergens når vi skal løse Monodomeneproblemet.

Vi går videre til å løse Monodomeneproblemet. Denne gangen skal vi løse problemet på et grid i 3D. Gridet blir satt til en boks på (20,7,3)mm, hvor vi har mulighet til å variere cellestørrelsen i boksen. Feilen er beregnet ved T = 10ms, og referanseløsningene er beregnet med dt = 0.0009765625. Tabell 7.26 viser feil og konvergens

dt	e	e_0/e_1
1	0.038523	
0.5	0.025520	1.5095
0.25	0.003956	6.4505
0.125	0.002678	1.4773
0.0625	0.000728	3.6773
0.03125	0.000457	1.5945
0.015625	8.327e-05	5.4851
0.0078125	2.424e-05	3.4350
0.00390625	5.924e-06	4.0916
0.001953125	1.205e-06	4.9182

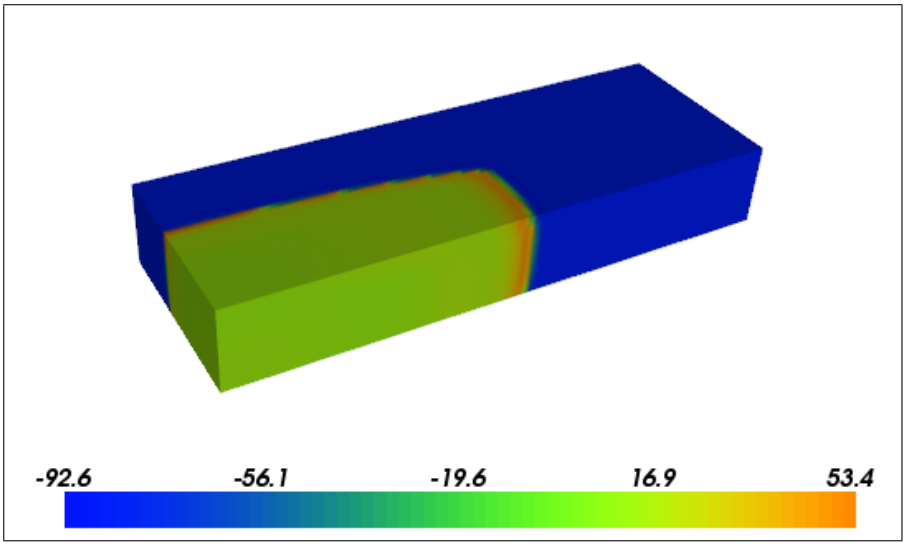
Tabell 7.25: Feil og konvergens for aksjonspotesialet i en celle, løst med GRL2

	Euler		CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1	e	e_0/e_1
1	0.726112		0.668261		0.665909	
0.5	0.694756	1.0451	0.550262	1.2144	0.543564	1.2251
0.25	0.631631	1.0999	0.414118	1.3288	0.432762	1.2560
0.125	0.534292	1.1822	0.256227	1.6162	0.293353	1.4752
0.0625	0.423237	1.2624	0.118711	2.1584	0.150939	1.9435
0.03125	0.322498	1.3124	0.039099	3.0362	0.052389	2.8811
0.015625	0.231748	1.3916	0.010571	3.6986	0.014096	3.7167
0.0078125	0.143714	1.6126	0.002714	3.8956	0.003588	3.9282
0.00390625	0.071985	1.9964	0.000661	4.1050	0.000869	4.1280
0.001953125	0.027248	2.6418	0.000134	4.9230	0.000176	4.9423

Tabell 7.26: Feil og konvergens for Monodomeneproblemet i 3D

for Monodomeneproblemet i 3D. For implisitt Euler har vi splittet med Godunov-splitting, for å bruke to 1.ordens metoder. Vi oppnår tilnærmet 1.ordens konvergens først når dt blir liten. CN og TR-BDF2 har begge blitt kombinert med Strang-splitting, dette for å opprettholde effekten av 2.ordens tidsdiskretisering. Det er også her nødvendig med en tilstrekkelig liten dt for å oppnå 2.ordens konvergens. I tillegg kan vi se at feilen i forhold til refranseløsningen blir minst ved CN som tidsdiskretiseringsskjema. Feilen mellom referanseløsningen gitt ved CN og refreranseløsningen gitt ved TR-BDF2 er i størrelseorden 1e-05, slik at løsningene konvergerer mot tilnærmet lik referanseløsning for både CN og TR-BDF2. Figur 7.8 viser hvordan løsningen vår ser ut etter T = 10ms.

Vi har her sett at vi trenger en dt tilstrekkelig liten for å oppnå ønskede konvergensrater. CN er i tillegg metoden som gir best nøyaktighet i forhold til referanseløsningen.



Figur 7.8: Løsningen av Monodomeneproblemet etter T = 10ms på vårt 3D grid

7.3.2 Aktiveringstid

Det er ønskelig å se på aktiveringstiden i gridet, og å se hvordan cellestørrelsen påvirker nøyaktighet. Aktiveringstiden for en node i gridet definerer vi som tidspunktet transmembranpotesialet passer 0. Når transmembranpotesialet passer 0 vet vi noden i gridet er aktivert. Vi skal se på aktiveringstiden for hele gridet, og sammenligne for ulike cellestørrelser. Vi ønsker å la en bølge propagere gjennom hele gridet, slik at alle nodene blir aktivert. Deretter sammenligner vi feilen og konvergensen for aktiveringstiden på vår 3D boks med cellestørrelsen satt til $cell_size = 0.2$mm og varierende dt. Program 10 viser hvordan vi kan lage en 3D boks med varierende cellestørrelse. I tabell 7.27 ser vi feil og konvergens for aktiveringstiden på vår 3D boks, diskretisert med både CN og TR-BDF2 og splittet med Strang-splitting. Skjemaene gir ganske like resultater i forhold til både nøyaktighet og konvergens.

Program 10 Kode for 3D boks med varierende cellestørrelse

```
from dolfin import Box
import sys
cell_idx = int(sys.argv[1])
box_dim = [0, 0, 0, 20, 7, 3]
cell_sizes = [0.5, 0.2, 0.1, 0.05]
cell_size = cell_sizes[cell_idx]
parts=[]
for i in xrange(3):
    parts.append(int(round(box_dim[i + 3]/cell_size)))
mesh = Box(*(box_dim+parts))
```

	CN		TR-BDF2	
dt	e	e_0/e_1	e	e_0/e_1
1	1.404320		1.391914	
0.5	0.615938	2.2800	0.526127	2.6456
0.25	0.217449	2.8326	0.244165	2.1548
0.125	0.061846	3.5160	0.080912	3.0177
0.0625	0.018809	3.2881	0.024783	3.2648
0.03125	0.005384	3.4933	0.006981	3.5500
0.015625	0.001447	3.7222	0.001856	3.7617
0.0078125	0.000373	3.8817	0.000475	3.9064
0.00390625	9.086e-05	4.1014	0.000115	4.1179
0.001953125	1.844e-05	4.9427	2.335e-05	4.9412

Tabell 7.27: Feil i ms og konvergens for aktiveringstiden til Monodomeneproblemet i 3D

Vi vil videre sammenligne aktiveringstiden på vårt referansegrid, beregnet med $cell_size = 0.2$mm, med et finere grid, hvor $cell_size = 0.05$mm. For å forenkle beregningene og korte ned på tidsbruken, så velger vi å sammenligne betydningen av cellestørrelse på et 2D grid, (20,7)mm. For ytterligere å holde tidsbruken nede, velger vi bare CN som tidsdiskretiseringsskjema. Tabell 7.28 viser feil og konvergens for aktiveringstiden, hvor $cell\_size$ er satt til henholdsvis 0.2mm og 0.05mm. For begge $cell_size$s får vi god nøyaktighet i forhold til referanseløsningene beregnet ved dt = 0.000976562. Vi er dermed avhengig av å sammenligne gridene bedre for å avgjøre hvor god celleoppløsning som er nødvendig. I figur 7.9 kan vi se aktiveringstiden i ms for vårt 2D grid med (a) $cell_size = 0.2$mm og (b) $cell_size = 0.05$mm. Vi kan tydelig se at vi får en jevnere propagering ved å velge et fint grid.

Dermed er det ønskelig å sammenligne gridstørrelser mer eksakt. Vi beregner aktiveringstiden på fire grid med ulik cellestørrelser: $cell\_size = 0.2$, $cell\_size = 0.1$, $cell\_size = 0.05$ og $cell\_size = 0.025$, alle gitt i mm. Disse gridene vil inneholde ulikt antall noder, så for å kunne sammenligne aktiveringstiden fra disse fire kjøringene må vi sørge for å få like mange noder, og dermed også likt antall verdier for aktiveringstid, på de grove gridene som vårt referansegrid med $cell_size = 0.025$mm. Måten å gjøre dette på vil være å interpolere. Det vil si at vi på et grovt grid danner nodepunkter som mangler i forhold til referansegridet, og beregner aktiveringstiden i disse nodepunktene ved hjelp av verdier for aktiveringstid i omkringliggende nodepunkter. Vi lar dt=0.0078125 og beregner så aktiveringstiden på de 4 gridene. Deretter interpolerer vi de 3 grove gridene mot vårt referansegrid gitt

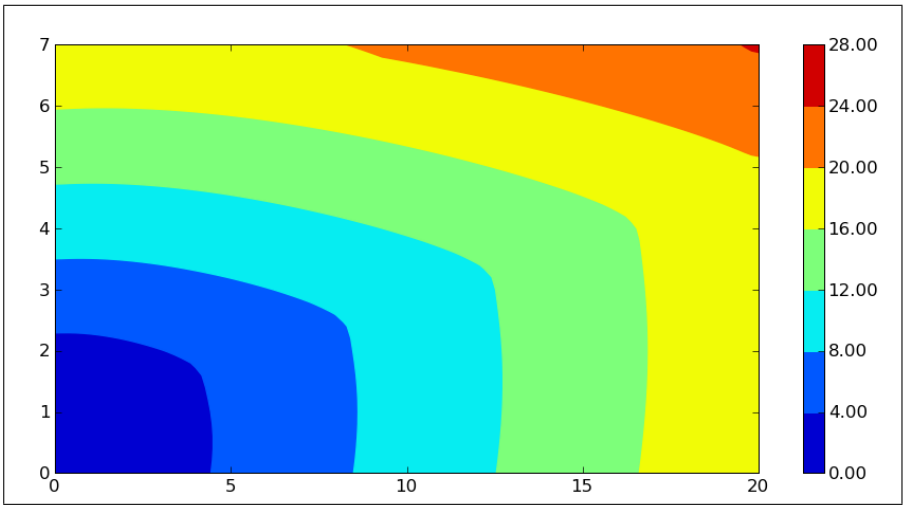
	cell_size = 0.2		cell_size = 0.05	
dt	e	e_0/e_1	e	e_0/e_1
1	1.437302		1.642456	
0.5	0.637099	2.2560	0.823654	1.9941
0.25	0.222070	2.8689	0.407267	2.0224
0.125	0.059741	3.7172	0.147464	2.7618
0.0625	0.018431	3.2413	0.042558	3.4650
0.03125	0.005274	3.4947	0.010218	4.1650
0.015625	0.001419	3.7163	0.002465	4.1539
0.0078125	0.000366	3.8815	0.000613	4.0233
0.00390625	8.912e-05	4.1023	0.000147	4.1658
0.001953125	1.808e-05	4.9299	2.981e-05	4.9332

Tabell 7.28: Feil i ms og konvergens for aktiveringstiden til Monodomeneproblemet i 2D løst med Strang-splitting og CN

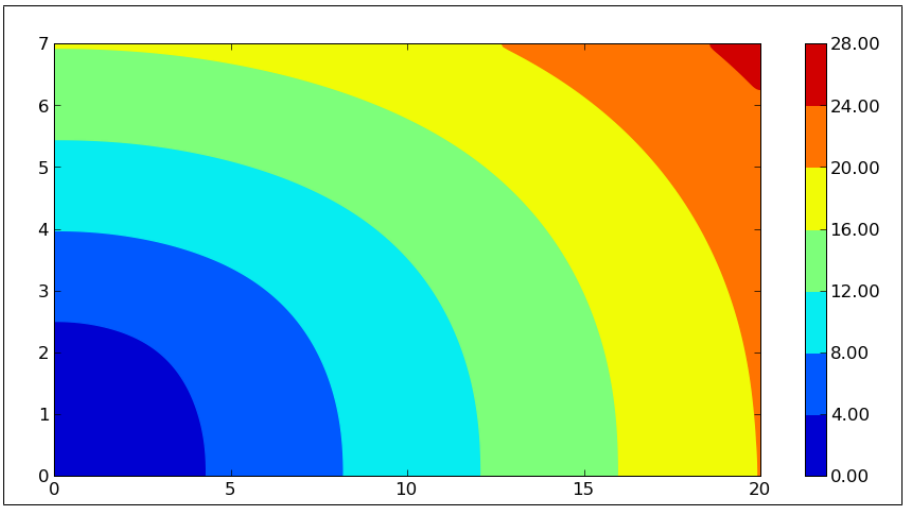
$cell\_size$	e
0.2	0.118624
0.1	0.103390
0.05	0.050765

Tabell 7.29: Feil i ms for aktiveringstiden for ulike cell_size

ved $cell_size = 0.025$mm. Dette gir oss muligheten til å sammenligne feil i aktiveringstid gitt av cellestørrelse i gridet. Tabell 7.29 viser feilen i ms for aktiveringstiden. Feilen er under 0.01ms for hele gridet for $cell_size = 0.05$mm i forhold til $cell_size = 0.025$mm, hvilket er så lite at det vil være tilstrekkelig å velge $cell_size = 0.05$mm for å spare simuleringstid.



(a) cell_size = 0.2



(b) cell_size = 0.05

Figur 7.9: Aktiveringstiden i ms for 2D grid

Kapittel 8

Konklusjon

Implisitt Euler, eksplisitt Euler og Crank-Nicholson er som regel det foretrukne valget for tidsdiskretisering av Monodomene- og Bidomenemodellen. Vi ville for denne oppgaven teste om en annen numerisk metode kunne gi bedre resultater med hesnyn på nøyaktighet og CPU-tid. Valget falt på TR-BDF2, som er en A-stabil og L-stabil metode, samt forholdsvis enkel å implementere til tross for at den består av 2 (indre) steg.

I seksjon 7.1.1 og seksjon 7.2.1 så vi på nøyaktigheten til de 3 metodene for henholdsvis Monodomene- og Bidomeneproblemet. Som forventet ga implisitt Euler dårligst nøyaktighet for løsningene. Dette skyldes at metoden er av 1.orden. Den krever dermed en mindre dt for å oppnå god nøyaktighet i forhold til en 2.ordens metode og vil med det bruke lenger CPU-tid. For de to andre metodene viste det seg at TR-BDF2 ga noe bedre nøyaktighet enn CN, hvor det var spesielt forskjell i nøyaktighet ved løsning av Monodomeneproblemet. For Bidomeneproblemet ga begge metoder god nøyaktighet og forskjellen var minimal. Det er verdt å merke seg at splittemetoden vi bruker bør være av samme orden, eller høyere, for å oppnå best mulig nøyaktighet. Strang-splitting ble derfor brukt.

I tillegg til å se på hvor nøyaktige de tre tidsdiskretiseringsmetodene våre er med hensyn på transmembranpotensiale, har vi også sett på nøyaktigheten ved beregning av propageringshastighet i seksjon 7.1.3. På lik linje som med transmembranpotensialet viser det seg at TR-BDF2 gir de mest nøyaktige løsningene for propageringshastighet. Propageringshastigheten kan i tillegg settes i sammenheng med upstroke velocity, som er et mål på hvor raskt transmembranpotensialet i cellene endres i depolariseringsfasen. Upstroke velocity kan variere avhengig av hvilken cellemodell som er valgt. Seksjon 7.1.4 viser hvilken effekt upstroke velocity kan ha på propageringshastighet og nøyaktigheten av løsningene. Stive cellemodeller har

en rask upstroke velocity, og våre resulateter viser da at vi er avhengig av en liten dt for å oppnå gode resultater. Ser vi på transmembranpotensialet i en enkelt celle, så ser vi at nøyaktigheten av løsningene endrer seg ved endret upstroke rate. En lav upstroke rate gir gode nøyaktige løsninger ved stor dt, mens en høy upstroke rate krever en mindre dt for å oppnå tilsvarende nøyaktighet. Dette gjenspeiles også når vi ser på løsningene av transmembranpotensialet og propageringshastigheten for Monodomeneproblemet. Dermed vil valg av cellemodell, og dens upstroke rate, være med på å avgjøre hvor liten dt vi trenger for å oppnå gode nøyaktige løsninger. Det kunne i tillegg sees i seksjon 7.1.5 at vi må ta hensyn til gridoppløsningen. I vårt tilfelle trengte vi en gridoppløsning i x-retning på $\Delta x < 0.05$mm for at propageringshastigheten skulle stabilisere seg. For stor Δx vil gi avvik i propageringshatigheten. I tillegg kunne vi risikere verdier for transmembarnpotensialet under v_{rest} (her satt til -85mV).

Valg av cellemodell, med sin upstroke rate, og ønsket nøyaktighet av løsningene legger en føring på hvor liten dt som er nødvendig. For å avgjøre hvilken numerisk metode som er ønskelig å benytte, må vi også legge CPU-tid til grunn for valget. Dette så vi på i seksjon 7.1.6 og seksjon 7.2.2 for henholdsvis Monodomene- og Bidomeneproblemet. En liten dt vil gi en beregning som er mer CPU krevende enn en stor dt. Hvis vi legger til grunn at vi bruker likningsløseren med prekondisjonering og $x_{start} = v_{n-1}$ kan vi se at forskjellen i tidsbruk, for Monodomeneproblemet, mellom de tre numeriske metodene øker etterhvert som dt blir mindre. For Bidomeneproblemet ga resultatene at forholdet i tidsbruk mellom de tre metodene var lik uavhengig av størrelse på dt. Implisitt Euler er en metode som krever en liten dt for å oppnå god nøyaktighet. CPU-tiden vil øke etterhvert som dt blir mindre, så valg av implisitt Euler som metode kan bli simuleringsmessig krevende, med hensyn på tidsbruk, hvis vi er ute etter en god nøyaktig løsning. Da vi så på nøyaktigheten av løsningene i seksjon 7.1.1 kom vi frem til at TR-BDF2 var noe mer nøyaktig enn CN. Legger vi i tillegg tidsbruken til grunn ved valget mellom disse to metodene, så vi at for en gitt dt ved TR-BDF2 kunne vi velge en dt halvparten så stor for CN uten å bruke nevneverdig mer tid. Dette gjaldt for både Monodomene- og Bidomeneproblemet. Dermed kunne vi hente inn fortrinnet TR-BDF2 har på nøyaktighet. Dette vil også være en fordel ved valg av cellemodell med rask upstroke velocity. Vi kan dermed konkludere med at CN bør være det foretrukne valget av disse tre numeriske metodene med hensyn på både nøyaktighet og CPU-tid.

I seksjon 7.3 gikk vi videre med Monodomeproblemet, denne gang kombinert med en mer realistisk cellemodell. Dette er en cellemodell med rask upstroke velocity,

og det var tydelig å se i seksjon 7.3.1 at det var nødvendig med en liten dt før vi oppnådde bedre nøyaktighet og gode konvergensrater. Ved bruk av Panfilov som cellemodell fikk vi gode konvergensrater allerede ved $dt = 2^{-3}$, mens for TNNP som cellemodell var det nødvendig med en $dt = 2^{-7}$. For valg av numerisk metode var det i tillegg tydelig at CN her ga de mest nøyaktige løsningene. I seksjon 7.3.2 så vi i tillegg på aktiveringstiden for hele domenet. Aktiveringstiden er et annet mål på hvor raskt propageringen skjer. Som et tidsbesparende valg ble CN brukt som tidsdiskretiseringsmetode da vi undersøkte betydningen av cellestørrelse. Vi så tidligere i oppgaven på betydningen av gridoppløsning ved beregning av propageringshastighet, hvor vi kom frem til en ønsket gridoppløsning i x-retning. Det vil være tidkrevende å velge liten cellestørrelse, men samtidig er det ønskelig med god nøyaktighet. Vi sammenlignet et grid gitt med fire ulike cellestørrelser: 0.2, 0.1, 0.05 og 0.025, gitt i mm. Feilen i ms for aktiveringstiden var på godt under 0.1ms for vårt grid med cellestørrelse 0.05mm i forhold til referansegridet med cellestørrelse 0.025mm. Dette vil gi en tilstrekkelig nøyaktighet, og vi kan dermed spare CPU-tid ved å velge 0.05mm som cellestørrelsen.

En liten del av oppgaven ble viet til likningsløseren og et par egenskaper ved denne. I seksjon 7.1.2 så vi på valg av toleranseverdi for feil i likningsløseren. En for liten TOL-verdi vil gi unødvendig lang CPU-tid. For implisitt Euler hadde det liten betydning om TOL ble satt til 1e-05 eller 1e-16. Dette henger sammen med at feilen for løsningen i forhold til referanseløsningen er stor. For CN vil TOL = 1e-05 gi feil som ikke følger 2.ordens konvergens, mens en mindre TOL-verdi vil gi ønskede resultater. TR-BDF2 derimot var avnhgig av en liten nok TOL-verdi. TOL = 1e-05, samt TOL = 1e-08 med små dt vil gi utslag for konvergensen av feilen for TR-BDF2. Samtidig så vi at feilen for Monodomeneproblemet med TR-BDF2 konvergerte mot tilnærmet lik referanseløsning ved TOL = 1e-08 og TOL = 1e-16, og vi kunne dermed sette TOL til 1e-08 for å spare beregninger og CPU-tid. I seksjon 7.1.6 og seksjon 7.2.2 så vi i tillegg på betydningen av prekondisjonering av likningssystemet og valg av startverdi. Det kom tydelig frem at prekondisjonering av systemet $Ax = b$ er nødvendig for å holde antall iterasjoner og CPU-tid nede, mens valg av initialverdi ikke var av like stor betydning om vi allerede hadde sørget for prekondisjonering av systemet. Spesielt for Bidomeneproblemet ga valg $x_{start} = 0$ meget høy CPU-tid og antall iterasjoner når vi ikke hadde brukt prekondisjonering. For Monodomeneproblemet var ikke variasjonene ved valg av initialverdi like store.

Tillegg A

Implementasjon

A.1 monodomainsolver.py

```
 #!/usr/bin/env python
"""This script demonstrate use of the classes
BidomainAssemblerAndSolver and IonicCellModels.
The monodomain problem is solved"""

__pyccdebug__ = 3

from BidomainAssemblerAndSolver import *
from py4c.IonicCellModels import *
from pycc.Functions import *
from viper import *
from py4c.Misc import *
from matplotlib import pylab
import time, os, sys


class monodomain_solver:
   def demo(self,cpu1=[],cpu2=[]):

      M=100; K=100;
      # Make the mesh
      mesh =  UnitSquare(M,K)
      n = len(mesh.coordinates())
```

```
# Specify conductivity properties
s_il = 3.0/2000
s_it = 0.3/2000
s_el = 2.0/2000
s_et = 1.4/2000
s_l = (s_il*s_el)/(s_il+s_el)
s_t = (s_it*s_et)/(s_it+s_et)

# A fixed tensor function, not spatially varying:
tf_fixed = Functions.FixedTensorFunction(
           numpy.array([s_l, s_t]))

# Set up the PDE problem:
pde = BidomainAssemblerAndSolver(self.dt, mesh,
      tf_fixed, self.method,self.use_TOL)
meshlist =  MeshLister(mesh, pde.mf,False)

# Specify ODE stuff
cellmodel =panfilov()
stim =[[(    1.0,    1.0),   0.0, 0.0,  0,  0.0]]
ode = IonicCellModels(n, cellmodel, meshlist, stim,
      GRL2(cellmodel))

# Set initial conditions for v
v = ode.initialTransmembranePotentials();
i = 0
for p in meshlist.get_nodes():
   x = p[0]
   v[i] = -85*(1-numpy.exp(-200.0*(x*x)))
   i += 1

# Plot value of v during simulation
if self.plot:
   pv = Viper(meshlist, v, -85,0)

# Time loop and solution algorithm
N = int(round(self.T/self.dt))
```

```
# Save solutions of v
if self.dump_solution == True:
   from py4c.Recorder import Recorder
   recorderV = Recorder(meshlist)
   sites = numpy.array([[0.0,0.5],[0.1,0.5],[0.2,0.5],
           [0.3,0.5],[0.4,0.5],[0.5,0.5],[0.6,0.5]])
   recorderV.selectCloseNodes(sites)
   recorderV.openResultFile('case'+"%d(T=%g).txt"
            %(self.a,self.T), N+1, self.dt);
   recorderV.dump(v)

# Solve with Godunov-split or Strang-split
if self.strang_split==False and self.cpu_print==False:
   for i in xrange(0, N):
      t = i*self.dt
      ode.forward(v, t, self.dt)
      pde.solve(v)
      if self.plot:
         pv.update()
      if self.dump_solution:
         recorderV.dump(v)

elif self.strang_split==False and self.cpu_print==True:
   for i in xrange(0, N):
      t = i*self.dt
      c0 = time.time()
      ode.forward(v, t, self.dt)
      self.cpu_ode += (time.time()-c0)
      self.cpu_pde += pde.solve(v)
      if self.plot:
         pv.update()
      if self.dump_solution:
         recorderV.dump(v)

elif self.strang_split==True and self.cpu_print==False:
   for i in xrange(0, (N)):
      t = i*self.dt
      ode.forward(v, t, 0.5*self.dt)
```

```
            pde.solve(v)
            ode.forward(v, t + 0.5*self.dt, 0.5*self.dt)
            if self.plot:
               pv.update()
            if self.dump_solution:
               recorderV.dump(v)

      else:
         c0 = time.time()
         ode.forward(v, 0, 0.5*self.dt)
         self.cpu_ode += (time.time()-c0)
         for i in xrange(0, (N-1)):
            t = i*self.dt
            self.cpu_pde += pde.solve(v)
            c0 = time.time()
            ode.forward(v, t , self.dt)
            self.cpu_ode += (time.time()-c0)
            if self.plot:
               pv.update()
            if self.dump_solution:
               recorderV.dump(v)
         self.cpu_pde += pde.solve(v)
         c0 = time.time()
         ode.forward(v, self.T - 0.5*self.dt, 0.5*self.dt)
         self.cpu_ode += (time.time()-c0)
         if self.plot:
            pv.update()
         if self.dump_solution:
            recorderV.dump(v)

      # Save solution
      name = "mono_dt(%g)_T(%d)_%s_%s" % (self.dt,self.T,
             self.strang_split,self.method)
      v.tofile(name)

      if self.plot:
         pv.interactive()
      if self.dump_solution:
```

```
            recorderV.closeResultFile()
        if self.cpu_print:
            pde.print_iter()
            cpu1.append(self.cpu_ode)
            cpu2.append(self.cpu_pde)

    def error(self,n):
        dt = 2**(-(n))
        name = "mono_dt(%g)_T(%d)_%s_%s" % (dt,self.T,
                self.strang_split,self.method)
        v_ref = numpy.fromfile(name)
        e = []
        print 'Referance given by solution with dt = ',dt
        print 'Error for monodomainproblem'
        print '(T=%g, method=%s)'% (self.T,self.method)
        print 'dt ', 'error'
        for a in range (n):
            dt = 2**(-a)
            name = "mono_dt(%g)_T(%d)_%s_%s" % (dt,self.T,
                    self.strang_split,self.method)
            u =numpy.fromfile(name)
            err = (v_ref-u)*(v_ref-u)
            err = math.sqrt(err.sum())
            err = err/math.sqrt((v_ref*v_ref).sum())
            print dt, err
            e.append(err)
        print 'Convergence'
        for a in range (0,(len(e)-1)):
            print  e[a]/e[a+1]

    def __init__(self,a, dt, T, strang, method,
                dump=False, plot=False, cpu_print=False):
        self.a = a
        self.dt = dt
        self.T = T
        self.strang_split = strang
        self.method = method
        self.dump_solution = dump
```

```
        self.plot = plot
        self.cpu_print = cpu_print
        self.cpu_ode = 0
        self.cpu_pde = 0
        self.use_TOL=1.0E-16

if __name__ ==’__main__’:
    # Variables
    T = 10
    strang = True
    method = ’trbdf2’
    dump_solution = True
    plot_solution = False
    calculate_cpu = False
    n=6

    cpu1 =  []
    cpu2 = []
    dtlist = []

    # Run monodomain_solver
    for a in range (n+1):
        dt = 2**(-a)
        dtlist.append(dt)
        ms = monodomain_solver(a,dt, T,strang,method,
             dump_solution, plot_solution, calculate_cpu)
        ms.demo(cpu1,cpu2)

    # Calculate error
    ms.error(n)

    # Print cpu-time
    for a in range (len(cpu1)):
        print ’dt =’,dtlist[a],’ cpu time for ode ’,cpu1[a]
        print ’dt =’,dtlist[a],’ cpu time for pde ’,cpu2[a]
```

A.2 BidomainAssemblerAndSolver.py

```
"""Sets up and solves the linear systems of the mono-
and bidomain equations.Uses Conjugate Gradients
method to solve the linear systems. """

from pycc import *
from pycc.LinearAlgebra import *
from pycc import Functions
from pycc.debug import debug

class BidomainAssemblerAndSolver:
   def __init__(self, dt, mesh, tf, method=cn, use_TOL=1.0E-8,
                torso_mesh=None, torso_tf = None):
      """Monodomain is used if torso_tf is absent. """
      self.iter = 0
      self.TOL = use_TOL
      self.prec = Prec # use the default pycc precondition

      self.monodomain = (torso_tf == None)
      if (not self.monodomain) and (torso_mesh == None):
          torso_mesh = mesh

      # Build matrices common for the bidomain and
      # monodomain problem
      debug("Building matrices", level=0)
      mf = MatrixFactory(mesh)
      self.M = mf.computeMassMatrix()
      self.Ai = mf.computeStiffnessMatrix(tf)
      self.mf = mf

      # Build different matrices depending on the time-
      # discretization
      self.method = method
      if self.method == 'trbdf2':
         print 'Method = trbdf2'
         gamma = 1 - math.sqrt(2)/2.0
         gamma2 = (1-(2*gamma))/(2*(1-gamma))
```

```
        self.A1 = self.M + gamma*dt*self.Ai
        self.b1 = self.M - gamma*dt*self.Ai

        self.A2 = self.M + gamma2*dt*self.Ai
        self.b2 = (1-gamma2)/(2*gamma)*self.M
        self.b3 = (1-(1-gamma2)/(2*gamma))*self.M
    else:
        if self.method == 'cn':
            print 'Method = crank-nicholson'
            a = 1.0; b = 1.0/2.0; c = 1.0; d = 1.0/2.0
        elif self.method == 'euler':
            print 'Method = implicit euler'
            a = 1.0; b = 1.0; c = 1.0; d = 0

        self.A = a*self.M + b*dt*self.Ai
        self.H = c*self.M - d*dt*self.Ai

    # Build matrices specific for each problem
    if (self.monodomain):
        if self.method == 'trbdf2':
            # self.P, self.P1 and self.P2 --> precond.
            self.P1 = self.prec(self.A1)
            self.cg1 = ConjugateGradients(self.A1, None,
                       self.P1,self.TOL, True, maxiter=
                       500, info=True)
            self.P2 = self.prec(self.A2)
            self.cg2 = ConjugateGradients(self.A2, None,
                       self.P2,self.TOL, True, maxiter=
                       500, info=True)
        else:
            self.P = self.prec(self.A)
            self.cg = ConjugateGradients(self.A, None,
                      self.P,self.TOL, True, maxiter=
                      500, info=True)
    else:
        print "Constructs bidomain matrices"
        mf_t = MatrixFactory(torso_mesh)
```

```
        self.MT = mf_t.computeMassMatrix()
        self.mf_t = mf_t
        Aie = mf_t.computeStiffnessMatrix(torso_tf)

        if self.method == 'trbdf2':
            # Create matrices
            B1 = gamma*dt*self.Ai
            B1t = gamma*dt*self.Ai
            C1 = gamma*dt*Aie

            B1.setNumCols(C1.m)
            B1t.setNumCols(C1.n)
            B1t.transposed = True

            B2 = gamma2*dt*self.Ai
            B2t = gamma2*dt*self.Ai
            C2 = gamma2*dt*Aie

            B2.setNumCols(C2.m)
            B2t.setNumCols(C2.n)
            B2t.transposed = True

            self.a1 = -gamma*dt*self.Ai
            self.a2 = -gamma*dt*Aie

            # Create the two block systems
            # self.Cc1, self.Cc2 -> precond.
            self.Ab1 = BlockMatrix((self.A1,B1),(B1t,C1))
            self.Cc1 = DiagBlockMatrix((self.prec(
                       self.A1),self.prec(C1)))
            self.Ab2 = BlockMatrix((self.A2,B2),(B2t,C2))
            self.Cc2 = DiagBlockMatrix((self.prec(
                       self.A2),self.prec(C2)))

            self.cg1 = ConjugateGradients(self.Ab1, None,
                       self.Cc1,self.TOL, True, maxiter=
                       500, info=True)
            self.cg2 = ConjugateGradients(self.Ab2, None,
```

```
                    self.Cc2,self.TOL, True, maxiter=
                    500, info=True)

        else:
            B = dt*self.Ai
            Bt = dt*self.A
            C = 2*dt*Aie

            B.setNumCols(C.m)
            Bt.setNumCols(C.n)
            Bt.transposed = True

            # Create the block system
            # self.Cc --> preconditioning
            self.Ab = BlockMatrix((self.A,B),(Bt,C))
            self.Cc = DiagBlockMatrix((self.prec(self.A),
                      self.prec(C)))
            self.H2 = -dt*self.Ai
            self.cg = ConjugateGradients(self.Ab, None,
                      self.Cc,self.TOL, True, maxiter=
                      500, info=True)

        debug("Done building matrices", level=3)

    def solve(self, x):
        from time import time
        start_time = time()
        if (self.monodomain):
            if self.method == 'trbdf2':

                vn = x.copy()
                x_guess = x.copy()
                # if x_guess = 0*x.copy() -> x_start=0 in CGM
                # if x_guess = x.copy() -> x_start=v_previous
                x_guess,info = self.cg1.solve(x_guess,
                               self.b1*x)
                x[:] = x_guess[:]
                self.iter += info['iter']
```

```
            b = self.b2*x + self.b3*vn
            x_guess = x.copy()
            x_guess,info = self.cg2.solve(x_guess,b)
            x[:] = x_guess[:]
            self.iter += info['iter']

        else:
            x_guess = x.copy()
            x_guess,info = self.cg.solve(x_guess,
                           self.H*x)
            x[:]=x_guess[:]
            self.iter += info['iter']

    else:
        if self.method == 'trbdf2':

            vn = x.copy()
            b =  BlockVector(self.b1*x[0]+self.a1*x[1],
                 self.a1*x[0]+self.a2*x[1])
            x_guess = x.copy()
            x_guess,info = self.cg1.solve(x_guess,b)
            x[1][:] = x_guess[1][:]
            x[0][:] = x_guess[0][:]
            self.iter += info['iter']

            b = BlockVector(self.b2*x[0]+self.b3*vn[0],
                0.0*x[1])
            x_guess = x.copy()
            x_guess,info = self.cg2.solve(x_guess,b)
            x[1][:] = x_guess[1][:]
            x[0][:] = x_guess[0][:]
            self.iter += info['iter']

        else:
            b = BlockVector(self.H*x[0], self.H2*x[0])
            x_guess = x.copy()
            x_guess,info = self.cg.solve(x_guess,b)
```

```
            x[1][:] = x_guess[1][:]
            x[0][:] = x_guess[0][:]
            self.iter += info['iter']

      return time() - start_time


   def print_iter(self):
      print 'Number of iterations:', self.iter
```

Bibliografi

[1] K. H. W. J. ten Tusscher; D. Noble; P. J. Noble; A. V. Panfilov. A model for human ventricular tissue. *Amercian Journal of Physiology - Heart and Circulatory Physiology*, 286(4):1573–1589, 2004.

[2] L.Tung. *A bidomain model for describing ischemic myocardinal D-C potentials*. PhD thesis, MIT, Cambridge, MA, 1978.

[3] W. Ying; C. S. Henriquez; D. J. Rose. Efficient fully implicit time integration methods for modeling cardiac dynamics. *IEEE Transactions on Biomedical Engineering*, 55(12):2701–2711, 2008.

[4] E. J. Vigmond; R. Weber dos Santos; A. J. Prassl; M. Deo; G. Plank. Solvers for the cardiac bidomain equations. *Progress in Biophysics and Molecular Biology*, 96(1-3):3–18, 2008.

[5] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 1968.

[6] Z. Qu; A. Garfinkel. An advanced algorithm for solving partial differential equation in cardiac conduction. *IEEE Transactions on Biomedical Engineering*, 46(9):1166–1168, 1999.

[7] M. Potse; B. Dube; J. Richer; A. Vinet; R. Gulrajani. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Transactions on Biomedical Engeneering*, 53(12):2425–2435, 2006.

[8] G. Fischer; B. Tilg; R.Modre; G. J. M. Huiskamp; J. Fetzer; W. Rucker; P. Wach. A bidomain model based BEM-FEM coupling formulation for anisotropic cardiac tissue. *Annals of Biomedical Engineering*, 28(10):1229–1243, 2000.

[9] J. P. Keener; K. Bogar. A numerical metrhod for the solution of the bidomain equations in cardiac tissue. *Chaos*, 8(1):234–241, 1998.

[10] R. E. Bank; W. M. Coughran Jr; W. Fichtner; E. H. Grosse; D. J. Rose; R. K. Smith. Transient simulation of silicon devices and cicuits. *IEEE Transactions on Electron Devices*, 32(10):1992–2007, 1985.

[11] M. E Hosea; L. F. Shampine. Analysis and implementation of TR-BDF2. *Applied Numerical Mathematics*, 20(1-2):21–37, 1996.

[12] J. Schewchuk. *An introduction to conjugate gradient method without the agonizing pain*. First edition, 1994.

[13] A. M. Katz. *Physiology of the Heart*. Lippincott Williams & Wilkins, third edition, 2001.

[14] R. M. Berne; M. N. Levy. *Physiology*. Mosby, fourth edition, 1998.

[15] N. A. Campbell. *Biology*. The Benjamin/Cummings Publishing Company, Inc., fourth edition, 1996.

[16] J. Sundnes; G. T. Lines; X. Cai; B. F. Nilsen; K. A. Mardal; A. Tveito. *Computing the electrical activity in the heart*. Springer, 2006.

[17] A. L Hodgkin; A. F. Huxley. A quantitative description of membrane current ant its application to conduction and excitation in nerve. *Journal of Physiology*, 117(4):500–544, 1952.

[18] R. A. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biopyhsical Journal*, 1(6):445–466, 1961.

[19] J. M. Rogers; A. D. McCulloch. A collocation-galerkin finite element model of cardiac actionpotential propagtion. *IEEE Transactions on Biomedical Engineering*, 41(8):743–757, 1994.

[20] R. R. Aliev; A. V. Panfilov. A simple two-variable model of cardiac excitation. *Chaos, Solitons & Fractals*, 7(3):293–301, 1996.

[21] W. Ying; C. S. Henriquez; D. J. Rose. Composite backward differentiation formula: an extension of the TR-BDF2 scheme. Submitted to Applied Numerical Mathematics, 2009.

[22] S. Rush; H. Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Transactions on Biomedical Engineering*, 25(4):389–392, 1978.

[23] J. Sundnes; R. Artebrant; O. Skavhaug; A. Tveito. A second order algorithm for solving dynamic cell membrane equations. *IEEE Transactions on Biomedical Engineering*, 56(10):2546–2548, 2009.

[24] M. R. Hestenes; E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.